Combining Neural Networks with Symbolic Approaches to perform Complex Event Processing on Non-Symbolic Data

Marc Roig Vilamala

A thesis submitted in partial fulfilment of the requirement for the degree of

Doctor of Philosophy (PhD)

July 2022

Cardiff University

School of Computer Science & Informatics

Abstract

This thesis presents three approaches to detecting situations of interest from non-symbolic data inputs such as images, audio and video through the use of Complex Event Processing (CEP) in an *agile*, *reliable* and *efficient* manner. These approaches must be agile, meaning that they must allow the implementation of solutions for a range of situations. We want them to be reliable, meaning that they must correctly detect the situations we are interested in. Finally, we want them to be efficient in terms of time and training data requirements.

First, we present ProbCEP, an approach that combines proxy models with symbolic programming to perform CEP. We consider proxy models consisting of pre-trained neural networks, which allow the system to use non-symbolic inputs. However, the data used to train such proxy models is not necessarily related to the situations of interest (called *complex events*) we want to detect. Logic rules are used to define under which conditions these complex events occur, based on the output of the proxy models. We also show how the speed of the system can be significantly increased using specific optimization techniques.

Then, we show two neuro-symbolic approaches, DeepProbCEP and Neuroplex. These approaches are designed to train a system to identify complex events from an input stream using small amounts of training data. Thanks to the injection of human knowledge into the system, these approaches require significantly less data than neural-only approaches.

Following that, we explore the reliability of DeepProbCEP against several adversarial attacks that poison the training data. We also demonstrate that DeepProbCEP is an agile approach, allowing users to change the behaviour of the system to adapt to many situations.

Finally, we discuss the potential of the research advances presented in this thesis for real-world applications, as well as possible areas for future research.

Acknowledgements

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Contents

List of Figures

List of Tables

List of Publications

The following articles present work relevant to this thesis where Marc Roig Vilamala, the submitting student, is the main author:

- [Roig Vilamala et al., 2019] Roig Vilamala, M., Hiley, L., Hicks, Y., Preece, A., and Cerutti, F. (2019). A pilot study on detecting violence in videos fusing proxy models. In *2019 22th International Conference on Information Fusion (FUSION)*, pages 1-8.

  This article presents ProbCEP, an approach designed to perform Complex Event Processing (CEP) using proxy models. These proxy models consist of neural networks pretrained using datasets that are not necessarily related to the complex events we are trying to identify. Most of Chapter 3 (excluding Section 3.7) describes the contents of this article.

- [Roig Vilamala et al., 2020] Roig Vilamala, M., Taylor, H., Xing, T., Garcia, L., Srivastava, M., Kaplan, L., Preece, A., Kimmig, A., and Cerutti, F. (2020). A hybrid neuro-symbolic approach for Complex Event Processing. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, 325.

  This extended abstract, published in Electronic Proceedings in Theoretical Computer Science (EPTCS) as part of the 36th International Conference on Logic Programming (ICLP 2020), presents DeepProbCEP, a neuro-symbolic approach to CEP capable of detecting complex events from an audio stream. The experiments performed in this publication are included in Section 5.2.4.

- [Roig Vilamala et al., 2021] Roig Vilamala, M., Xing, T., Taylor, H., Garcia, L., Srivastava, M., Kaplan, L., Preece, A., Kimmig, A., and Cerutti, F. (2021). Using DeepProbLog to perform Complex Event Processing on an audio stream. *StarAI*.

  This paper expands on the results from the extended abstract above by performing further experiments to evaluate DeepProbCEP's robustness against random noise in the training data. This allowed us to demonstrate that DeepProbCEP is robust against this type of adversarial attack. These experiments are discussed in Section 5.1.

- [Roig Vilamala et al., 2022] Roig Vilamala, M., Xing, T., Taylor, H., Garcia, L., Srivastava, M., Kaplan, L., Preece, A., Kimmig, A., and Cerutti, F. (2022). DeepProbCEP: A neuro-symbolic approach for complex event processing in adversarial settings. *Expert Systems with Applications*. Under review.

  This journal paper, currently in the second stage of the review process for Expert Systems With Applications (ESWA), further expands the experiments performed on DeepProbCEP and other complex event processing approaches. This includes more evaluations of robustness against adversarial attacks for DeepProbCEP (which are also discussed in Section 5.1). It also includes experiments evaluating the agility of DeepProbCEP, which are included in Section 5.2. Finally, it performs experiments to evaluate how between different state-of-the-art complex event processing approaches perform when training with sparse data, which has become the main focus of Chapter 4 in this thesis.

Meanwhile, the submitting student is the second author in the following articles, which also present work relevant to this thesis:

- [Xing et al., 2019] Xing, T., Roig Vilamala, M., Garcia, L., Cerutti, F., Kaplan, L., Preece, A., and Srivastava, M. (2019). DeepCEP: Deep complex event processing using distributed multimodal information. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 87-92.

  This article presents DeepCEP, a neuro-symbolic approach to complex event processing developed by fellow PhD student Tianwei Xing with our collaboration. An updated version of this approach (see the following article) is discussed in the thesis.

- [Xing et al., 2020] Xing, T., Garcia, L., Vilamala, M. R., Cerutti, F., Kaplan, L., Preece, A., and Srivastava, M. (2020). Neuroplex: Learning to detect complex events in sensor networks through knowledge injection. *18th Conference on Embedded Networked Sensor Systems*, pages 489-502.

  This article presents Neuroplex, a neuro-symbolic approach to CEP that improves on DeepCEP by allowing the end-to-end training of the neural network used as a perception layer. Neuroplex is described in Section 4.3 and compared against other approaches throughout Chapter 4.

Chapter 1

Introduction

Complex Event Processing (CEP) systems are designed to identify situations of interest (called *complex events*) from an input stream of atomic data (where each atomic piece of information is called an *event*, or *simple event*). Such systems can be effective for situational understanding, as they can allow for the automatic identification of specific situations, as defined by the user. Situational understanding requires both insight and foresight. In its traditional definition [Dostal, 2007] it is the "product of applying analysis and judgement to the unit's situation awareness to determine the relationships of the factors present, and form logical conclusions concerning threats to the mission accomplishment, opportunities for mission accomplishment, and gaps in information." [Ministry of Defence UK, 2016] goes further, explicitly mentioning that (situational) "understanding involves acquiring and developing knowledge to a level that enables us to know why something has happened or is happening (insight) and be able to identify and anticipate what may happen (foresight)."

Traditionally, CEP systems were designed to detect complex events from streams of symbolic data [Robins, 2010, Teymourian et al., 2012, Skarlatidis et al., 2015]. This was done through a manual definition of rules, which specified under which conditions a complex event happened, thus allowing the systems to automatically identify when complex events occurred based on the simple events from the input. These rules are traditionally defined by an expert in the subject being detected, who should be able to define the situations they are interested in (the complex events) based on a combination of atomic pieces of data (the simple events). These approaches have shown great potential when being used on symbolic data. However, the fact that users need to specify rules to process the input data makes it infeasible to use them for certain types of inputs, such as images, audio or video. This is because, while theoretically possible, manually defining rules to identify the content of an image, audio or video would be tedious and inefficient. For the purpose of this thesis, we will refer to such types of data for which rules cannot (easily) be manually defined to extract information as *non-symbolic data*.

In this thesis, we present approaches that are able to detect complex events from non-symbolic data inputs. Since the input data cannot be processed directly using symbolic rules, our approaches combine the symbolic approaches with systems capable of automatically extracting symbolic information from the non-symbolic data. This is done through the use of neural networks, which classify the non-symbolic data into a pre-defined set of classes. This allows the user to define rules that identify the complex events based on the given set of classes.

The difficulty of this approach, however, is the fact that complex events tend to be, by nature, relatively rare. This means that it can be difficult to get access to enough training data for such neural networks to learn. As a result, traditional machine learning approaches, such as neural networks, can struggle to properly learn how to detect the complex events on their own, as they tend to require larger amounts of training data.

It is also difficult to train the neural networks on their own to then combine them with the symbolic approaches, as obtaining labels for the simple events is significantly more time-consuming than just labelling when the complex events occur. This is because there is a much larger number of simple events for the same amount of input data. Furthermore, the occurrence of simple events may not be as clear to the people labelling as the complex events we are trying to detect. As such, obtaining labels for the simple events to train the neural network on its own can be significantly more expensive, and thus not the best solution.

The first approach we propose, ProbCEP, aims to avoid this issue of obtaining training labels by using pre-trained proxy models instead. This allows the use of models that have been pre-trained on data that does not necessarily include the situations we are trying to detect, thus eliminating the issue of finding enough training data. As such, this approach can easily be implemented for many scenarios, allowing for quick deployment. The ProbCEP approach uses a symbolic layer in which the user is able to define rules that describe when the complex events happen based on the output of the proxy models.

For our other approaches, we explore how neuro-symbolic architectures allow users to inject human knowledge into the system, reducing the complexity of the learning process and making it possible to train with significantly less data. Neuro-symbolic artificial intelligence is an area of research with growing interest that combines traditional rule-based approaches with modern deep learning techniques, with the objective of offering the benefits of both approaches [De Raedt et al., 2019]. More specifically, it combines the low-level perception capacities from neural approaches that allow them to deal with non-symbolic data while retaining the high-level reasoning benefits from symbolic approaches. In the area of CEP, neuro-symbolic approaches offer several advantages when compared to *neural-only* approaches (that is, approaches that rely solely on conventional neural networks). Firstly, neuro-symbolic approaches have been shown to be able to learn with significantly less data [Susskind et al., 2021], which reduces the issue of obtaining enough training data. Furthermore, CEP approaches have traditionally been implemented using symbolic approaches, by defining a set of conditions for each complex event. Then a complex event is considered to happen when the corresponding conditions are fulfilled. As such, neuro-symbolic approaches are a good choice when using non-symbolic data, as they allow users to retain manual definitions for the complex events through the symbolic layer, while the neural layer allows the system to process the non-symbolic data. Of course, this is only the case if experts are available to manually define the symbolic rules that specify when a complex event occurs. For the purpose of this thesis, we assume that we have symbolic rules required for the complex events that we are interested in, and that those are correct. If such rules were not available or, even worse, they were incorrect, the proposed approaches might provide no advantage or even harm the performance of the system.

It is worth noting the difference between the ProbCEP approach and the neuro-symbolic approaches. While both combine neural networks with a symbolic approach, the steps taken to prepare the system to predict when complex events are happening are significantly different. ProbCEP uses pre-trained proxy models. As such, training is not a part of this process. Instead, users need to define how the outputs of the proxy models can be combined to detect the complex events they are interested in. In contrast, neuro-symbolic approaches focus on training the system so that it learns to detect complex events.

In this thesis, we explore two different neuro-symbolic approaches to CEP. First, we explore DeepProbCEP, which was developed mainly at *Cardiff University*. DeepProbCEP uses a probabilistic logic layer to allow the system to train a low-level perception neural network, which extracts the symbolic information from the non-symbolic input data. We also explore Neuroplex, developed at the *University of California, Los Angeles* in collaboration with *Cardiff University*. Neuroplex uses a neural network to emulate the functionality of the logic layer, which allows it to backpropagate through this logic layer and train the low-level perception neural network. As such, the main difference between DeepProbCEP and Neuroplex is in how they implement the logic layer in a manner that allows for the training of the low-level perception neural network.

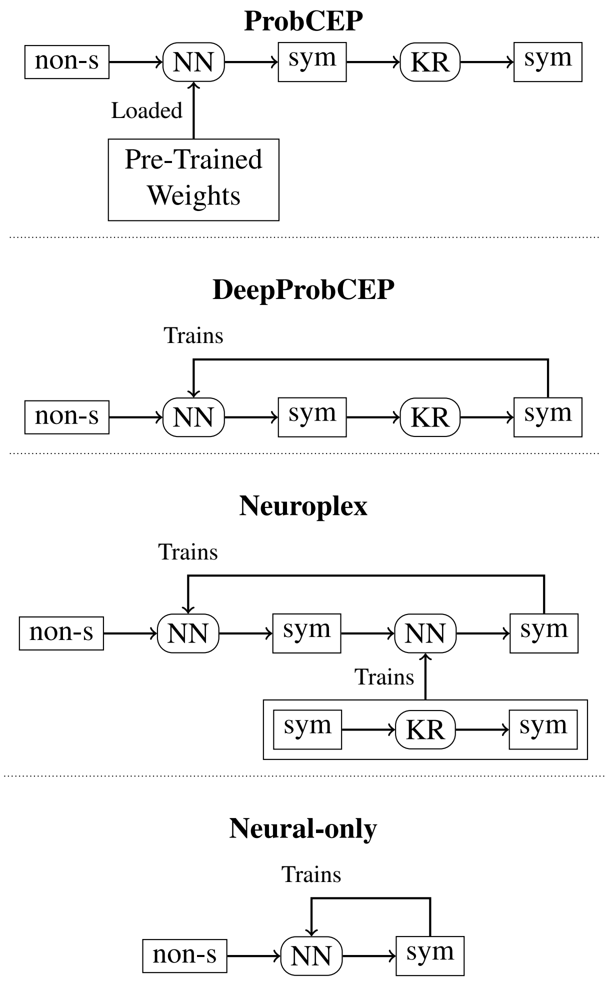


Figure 1.1: General schematic of the approaches considered in the thesis. Rectangular boxes represent data, using *sym* for symbolic data and *non-s* for non-symbolic data. Oval boxes represent algorithms, using *NN* for neural network components and *KR* for knowledge reasoning (symbolic) components.

Figure 1.1 shows a simplified architecture of the four types of approaches considered in this thesis, allowing us to visualize the differences between the approaches. These differences can change which of the approaches is most appropriate to solve a certain circumstance, as we explore in the rest of the thesis.

It is worth noting that all the considered approaches work in an *end-to-end* manner, meaning that they are able to generate their outputs (that is, when complex events are happening) based on the non-symbolic input (that is, the images, audio or video from which the users want to detect complex events). While Figure 1.1 does show intermediate stages for most of the approaches, these are simply shown to separate two sub-systems for the same approach. Similarly, all the approaches that train the system do so using *end-to-end training*, meaning that all the training points provided to the systems consist only of non-symbolic inputs and the corresponding complex event prediction, without any labels for the intermediate stages.

1.1 Motivating scenario

Our world has a large number of devices able to constantly detect many different sources of information. For instance, London has a large amount of CCTV cameras, and almost everyone carries a microphone in the shape of a smartphone. These sources of information, along with many others, could be used to detect situations of interest very quickly. For instance, CCTV cameras could be used to detect fights in the streets or traffic accidents. This could allow for a quicker response by the relevant authorities. The difficulty, however, is in detecting these situations from the streams of information that these sensors provide. Of course, humans are able to identify these situations with relative ease. However, it is not feasible for humans to process all the streams at the same time. Instead, our objective in this research is to create a system capable of reducing the amount of information that has to be manually processed by trying to automatically identify which situations may be of interest. From a CEP perspective, the CCTV cameras, microphones and other similar sources would be the *input streams of atomic data*. Meanwhile, the fights, traffic accidents and other situations of interest we can detect from such streams would be the *complex events* we are aiming to identify. In order to identify the complex events, we rely on detecting pre-defined combinations of *simple events*, which are obtained from the input streams. Simple events are considered instantaneous or to take a very short period of time relative to the length of complex events. For example, we might say that the complex event *fight* happens if multiple simple events *punch* occur in a certain window of time. The number of *punch* simple events and the length of the window of time would be determined by experts, based on their previous knowledge. This thesis explores how we can define which combinations of simple events result in which complex events, as well as how to identify the simple events from the streams of data (e.g. how to detect that a punch happens from a CCTV feed). In this section, we explore a motivating scenario based on an audio input.

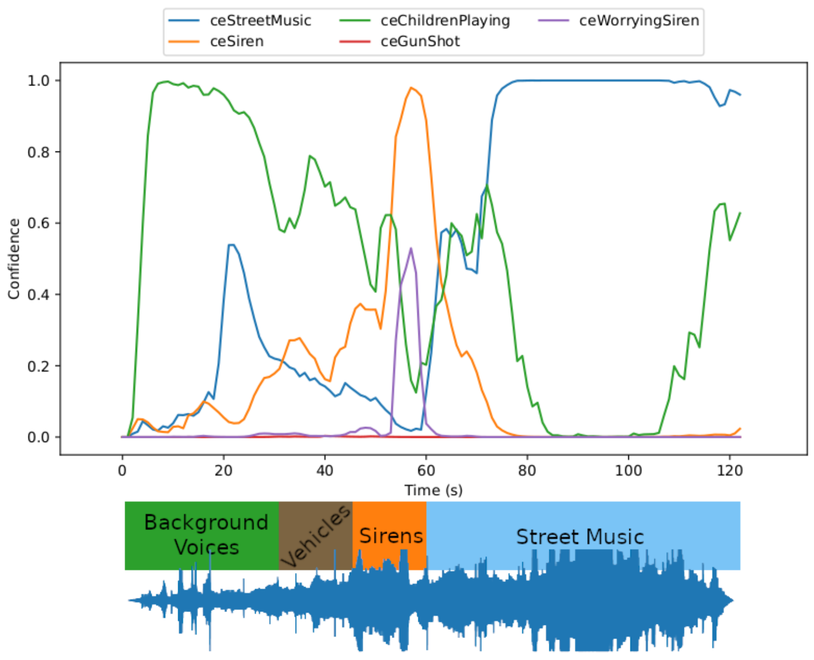
Figure 1.2 shows an example of what a system created through this research could do. The graphs show the predictions made by the system for each second of the input files, with higher values indicating a higher likelihood that the corresponding situation of interest is occurring[1]. Figure 1.2a shows the results for a recording taken by someone crossing a busy street[2]. As indicated by the box under the graph, the audio starts with background voices until around 30 seconds into the file. This is followed by the sound of vehicles. After, the sound of sirens approaching and getting louder can be heard between seconds 45 and 60. Finally, street music can be heard until the end of the file. Figure 1.2b shows the results for a modified version of the same file. The first fragment of the file (about 60 seconds long) remains the same, up to the point at which we cannot hear the sirens anymore. However, in this modified version, the sound of street music has been edited out and substituted by the sound of gunshots[3]. For the purpose of this example, the sounds heard at each point can be considered simple events. More specifically, we consider the main sound heard at every second of the recording to be a simple event.

The system used is able to detect several complex events in the given audio files. Most of these consist simply of the presence of a specific sound in the audio. For instance, the system detects the sound of children playing at the start of both files. This is indicated by the complex event *ceChildrenPlaying*, which is defined to be multiple instances of the simple event *children playing* in a row. While the background voices seem more likely to be those of adults, this confusion is understandable as children playing is by far the closest class the system has been trained to recognize. Similarly, the system is able to recognize the sirens (*ceSiren*) in both

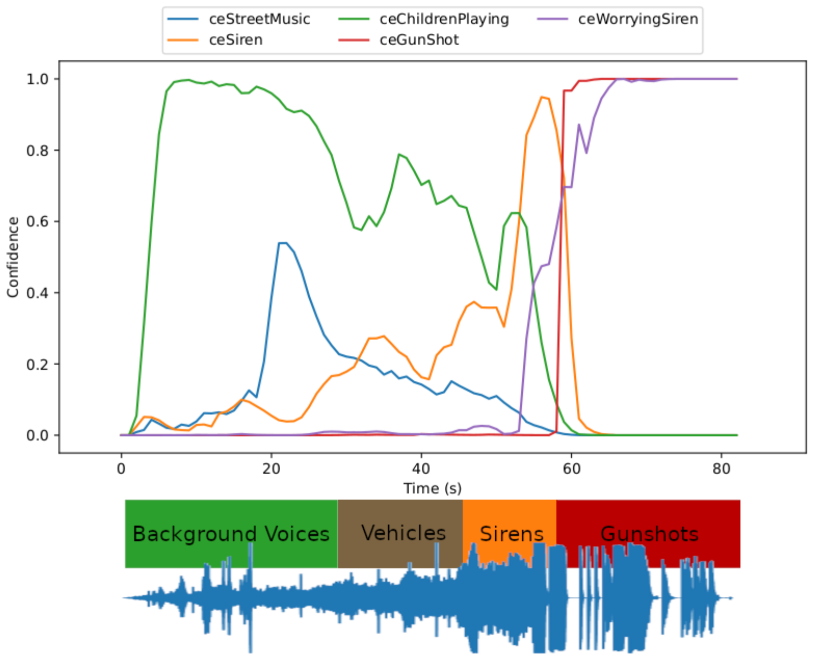
[1]A video showing the predictions as the audio is playing can be found at `https://www.youtube.com/watch?v=dllH0VzppPM` (on 13th of April 2022).

[2]The original audio can be found at `https://freesound.org/people/bulldozia/sounds/106905/` (on 13th of April 2022)

[3]The sounds of gunshots used for this modification consist of a combination of `https://freesound.org/people/tcpp/sounds/151556/` and `https://freesound.org/people/Pashee/sounds/415508/` (on 13th of April 2022)



(a) Original audio file



(b) Modified audio file with gunshots

Figure 1.2: Predictions for two audio files. The boxes between the waveforms and the graphs show the main sound present in the audio for each period of time.

files, as well as the street music (*ceStreetMusic*) and gunshots (*ceGunShot*) from each of the files. Each of these are defined as the corresponding simple event (*siren*, *street music* and *gunshot* respectively) happening multiple times in a row. The fact that any of these complex events is occurring signifies that the corresponding sound is persistent, instead of an isolated case. A more interesting complex event, however, is the predicted value for *ceWorryingSiren*. Roughly speaking, this complex event aims to identify when a siren is heard in a situation that may be concerning. This is done by detecting if there is an initially safe environment, followed by the sound of sirens and the disappearance of the sounds that made us think that the environment was safe. For this example, we consider children playing and street music to be safe environments, as they would seem to indicate that nothing wrong is happening in the area. Hearing sirens in such an environment may be concerning, but they could just be passing by, without anything happening in the area. This is, presumably, what happens in the original audio. Therefore, as shown in Figure 1.2a, the confidence in *ceWorryingSiren* quickly decreases as we start hearing the sounds of street music. However, if we started to hear more concerning sounds after those sirens, our confidence in the fact that something is happening in the area would increase. This is what happens in the modified version of the audio file, where gunshots start happening immediately after the sound of sirens is heard. As we would expect in this situation, the confidence in *ceWorryingSiren* quickly increases as the gunshots are heard, as shown in Figure 1.2b. This is because the *ceWorryingSiren* complex event is defined as the sequence of safe sounds (children playing or street music), followed by a siren and subsequently followed by more concerning sounds (gunshots).

It is worth noting that, thanks to how the complex events have been defined, the system also allows users to identify different situations, allowing them to organize the most appropriate response for each case. For instance, if no sirens had been detected before the gunshots, this would have only triggered *ceGunShot*, which may require a different response than *ceWorryingSiren*. As we will show in the rest of the thesis, our approaches are designed in a manner that allows users to specify exactly under which conditions each complex event should be triggered. This makes it possible for users to differentiate slightly different situations, thus allowing them to give the appropriate response in each case.

1.2 Research Question

In the field of complex event processing, multiple approaches make use of symbolic data to detect situations of interest. However, only a few approaches have been developed to work with non-symbolic data. This is because, while theoretically possible, it is usually infeasible to manually define rules to process the non-symbolic data. As such, additional parts need to be added to the CEP system to allow it to extract the symbolic information from the non-symbolic data. Other approaches decide to disregard the use of manually defined rules, using an approach based solely on training neural networks. In Section 2.3 we explore the different existing approaches to CEP that allow the use of non-symbolic data, as well as their limitations. With the intent of overcoming these limitations, we have developed two distinct approaches, ProbCEP and DeepProbCEP. We have also collaborated significantly in the development of Neuroplex, an approach designed by our colleagues from the University of California, Los Angeles. These approaches combine neural networks with symbolic programming with the intent of detecting complex events from a non-symbolic data stream. As such, our research question is:

"How does the combination of neural networks with symbolic programming improve the degree to which complex event processing can be performed on non-symbolic data in an agile, reliable and efficient manner?"

According to [Khan and Curry, 2020], an efficient processing of multimedia events is a key enabler for real-time and complex decision making in streaming media. In their paper, the authors suggest combining neural networks with symbolic approaches to solve this problem. They focus on video as their example, but we believe that such approaches also have promise in other areas where the input data is non-symbolic. However, as the authors highlight, there are also some challenges with such approaches, which we have needed to consider for ProbCEP, DeepProbCEP and Neuroplex:

1. **Suitable representation for reasoning.** A generalized and scalable model is required to allow for the representation of events and complex events.

2. **Expressive query definition and matching.** Approaches need to allow users to write high-level queries in an expressive and human-friendly manner.

3. **Labelling and training samples of event relations.** Learning each of the events and how they may relate to each other can be a difficult task, as there may be a large number of events during any given period of time. This can lead to difficulties annotating all the possible relations and having balanced datasets for all the relations.

4. **Consistent integration of knowledge bases.** Difficulties may arise when multiple knowledge bases need to be integrated, as the labels used by each of them may not match.

5. **Supporting rare or unseen relations.** Approaches must be capable of dealing with relations for which only small amounts of training data can be captured, or even none.

6. **Temporal processing of objects and relations.** Approaches must allow for the decomposition of the input feed into the relevant events, as well as allowing for an evaluation that takes into account the temporal aspect.

These challenges, among other considerations, have been included in the three adjectives from the research question: *agility*, *reliability* and *efficiency*. Throughout the thesis, we consider how our proposed approaches fulfil the requirements represented by each of these adjectives. In the following sections, we discuss the challenges represented by each of the adjectives, as well as any other considerations for that adjective.

1.2.1 Agility

agility is defined as the ability to move quickly and easily. As such, we want our approach to be able to quickly and easily change in order to adapt to new and changing environments. At the same time, we also want our approach to be able to make use of previously developed systems, in order to accelerate the rate at which progress can be made and reduce the amount of duplicated work needed. Therefore, we want an approach that produces systems that can be easily modified at any point, and that has the capabilities of representing any complex event definition the users want to express.

In regards to the challenges, this adjective covers part of Challenge 1. Specifically, the part where approaches need to provide a generalized representation. It also includes Challenge 2, as the query definitions need to be expressive enough to allow for the definition of any complex event required by the user, and easy enough to write and understand that modifications can be done quickly. Finally, it also includes Challenge 6, as the approaches should allow the user to choose which tools they want to use to decompose the input feed into events and their relations, as well as the types of events and relations they are considering.

In summary, approaches should make it easy for the user to define how they want to process the input feeds in order to detect the complex events, without limiting their expressivity. They should also make it possible to modify this processing if the users want to.

1.2.2 Reliability

Reliability is the quality of being trustworthy and performing consistently well. In the context of our research, we want our system to be reliable on two fronts. Firstly, we want a system that behaves as expected, and that can therefore be trusted. In the context of complex event processing, this means that the system should detect when complex events are happening if and only if the conditions we have defined for a complex event are met. This also includes Challenge 4, as the approach should not function incorrectly due to inconsistencies between different knowledge bases.

Secondly, a reliable system also needs to be robust. Robustness is the ability of a system to continue to operate correctly across a wide range of operational conditions and to fail gracefully outside of that range [Gribble, 2001]. As defined in current AI ethical principles [Board, 2019], approaches must be robust against adversarial conditions. In order to evaluate the robustness of the systems generated using our approaches, we will evaluate how they perform under adversarial attacks. In particular, we will focus on attacks against the training data, or more specifically the training labels. We decided to focus on these types of attacks because, depending on the types of complex events being considered, the people labelling the data may have differing opinions on the exact exact conditions that cause the complex event, possibly leading to inconsistent labelling. Evaluating the performance of the approaches against other types of adversarial conditions is left as future work.

1.2.3 Efficiency

Efficiency is defined as the good use of time and energy in a way that does not waste any. In the context of this research, we will be looking at mainly two types of efficiency: (i) **time-efficiency**, meaning that the system performs the task we want it to perform quickly and (ii) **data-efficiency**, meaning that the system can be trained with small amounts of data.

For time-efficiency, we will focus on the time required to train and prepare the systems, as well as the speed at which they can be used to detect complex events once ready. Time-efficiency also includes the part of Challenge 1 that was not covered by agility. In particular, approaches need to be efficient enough to allow for scalability.

For data-efficiency, we will focus on the amount of training data required to train the system. As complex events are relatively rare, a good data-efficiency can be necessary for systems to be able to train in certain situations. This covers Challenge 3, as reducing the need for training data also reduces the difficulties of annotating it. It also covers Challenge 5, as a more efficient use of training data may also allow us to learn rare relations.

1.3 Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 gives an overview of the literature by introducing the different types of complex event processing approaches. We perform a critical analysis of the existing approaches, evaluating their advantages and limitations. We also introduce the tools used for the development of the systems we present in this thesis, as well as an overview of the alternatives that could have been used instead.

Chapter 3 focuses on ProbCEP, a system designed to be *agile* and *efficient*. This chapter explains the ProbCEP approach, which uses pre-trained neural networks as a proxy model to detect complex events from an input stream through the use of manually defined symbolic rules. ProbCEP allows for the re-use of neural networks, which makes it possible to deploy new systems quickly as new environments are encountered. The outputs of these proxy models are evaluated using manually defined rules, which can be specified to detect the desired complex events. In Chapter 3, we evaluate the potential of ProbCEP by using it to detect violence from CCTV feeds.

Chapter 4 focuses on neuro-symbolic approaches instead. These are systems that combine the advantages of neural networks with symbolic approaches, making it possible to deal with non-symbolic data while using only small amounts of training data. In Chapter 4 we show that this is the case for two neuro-symbolic approaches: DeepProbCEP and Neuroplex. This demonstrates that neuro-symbolic approaches are *efficient* in terms of training data, making them a good choice for situations in which obtaining enough training data is difficult. This is quite relevant for CEP approaches, as complex events tend to be relatively rare. Chapter 4 also explores the *reliability* of the neuro-symbolic approaches, in terms of performing as expected.

The first half of Chapter 5 further explores the *reliability* of DeepProbCEP, specifically in the context of *robustness* against adversarial poisoning attacks. In the second half of Chapter 5, we explore how DeepProbCEP is an *agile* approach, allowing the user to change the behaviour of the system quickly and easily. As we demonstrate, this can be done in several ways, ranging from allowing different definitions for the complex events, to allowing changes in the behaviour of the system after it has been trained and even allowing multiple types of input data.

Finally, Chapter 6 discusses the key contributions of this thesis, as well as returning to the research question and motivating scenario shown in this chapter. This is followed by a discussion on the limitations of our approaches, before outlining future work and giving our final conclusions.

On top of that, there are appendices discussing other work that is related but not integral to the thesis. This includes the use of the PreCompilation library (Appendix A), which was developed to improve the time-efficiency of logic inference in ProbLog, the code for the sequence framework (Appendix B), which is used in the thesis but too long to include in the main text and results on performing poisoning attacks using audio data (Appendix C). There is also a glossary (Appendix D) with a description for the terms used throughout this thesis.

1.4 Summary of Contributions

The contributions shown in the rest of the thesis can be summarized as the following points:

- We present ProbCEP, an approach to CEP that allows the use of non-symbolic inputs without the need for training data.
- We also introduce the PreCompilation approach, which significantly increases the speed at which large numbers of similarly defined queries can be performed.
- We also discuss DeepProbCEP and Neuroplex, neuro-symbolic approaches that can learn to perform CEP on non-symbolic data with significantly smaller training datasets than neural-only approaches.
- Furthermore, we demonstrate that DeepProbCEP is a reliable and agile approach, providing a consistent performance in different scenarios.

Chapter 2

Background

2.1 Complex Event Processing

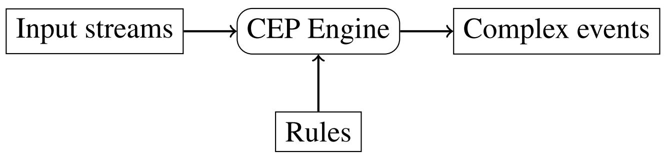


Figure 2.1: Simplified view of the behaviour of a CEP engine.

Complex Event Processing (CEP) is a technology that allows for the derivation of conclusions from the events present in a stream of information [Burgueño et al., 2018]. In order to derive these conclusions, CEP allows for the definition of complex events based on the events produced by the input streams, which allow the system to identify the situations of interest—that is, the complex events— automatically. Figure 2.1 shows a simplified view of a CEP engine. This simplified view is based on the structure proposed by [Cugola and Margara, 2012]. The *input streams* are fed into the CEP engine, which is then able to output the *complex events*. These complex events are the situations users are interested in from the input stream. The CEP engine is able to detect such complex events based on a set of rules, which are usually defined by experts in the field that is being detected. For instance, an expert on how fire spreads can define *rules* that specify how the change in the temperature detected by sensors (the *input stream*) can detect a when and where a fire is starting (the *complex event*). A CEP engine would then be able to automatically detect a fire starting based on those rules and the data provided by those temperature sensors. The process through which the user defines these complex events can change significantly depending on the exact approach being used, ranging from graphical tools [Roldán et al., 2020], to SQL-like languages [Bezerra et al., 2021, Burgueño et al., 2018], logic rules [Skarlatidis et al., 2015] and even languages specifically designed to define complex events [Chapnik et al., 2021, Yankovitch et al., 2022]. However, despite the varied implementations and the changes in the specifics, all CEP approaches generally follow the same principles where a complex event is defined based on combinations of events. According to [Luckham, 2002], an *event* is an object that can be subjected to computer processing and it signifies, or is a record of, an activity that has happened. Each *event* has three main aspects:

- *Form*: the form of an event is an object with particular attributes or data components, for instance, the time period of the activity;
- *Significance*: an event signifies an activity, hence an event's form usually contains data describing the activity it signifies;
- *Relativity*: an activity is related to other activities. Events have the same relationships with one another as the activities they signify. The *relativity* of an event refers to the set of relationships between that event and other events. An event's form usually encodes its relativities, i.e., methods to reconstruct the relationships with other events.

It is therefore important to notice that an event is not just a message or a record of an activity: the forms of events may be messages, but the events also have significance and relativity. In particular, the three main partial, transitive, and antisymmetric relationships between events are:

- *Time*: a relationship that orders events. While it relies on timestamps, in general, we need to be open to the possibility of uncertainty associated to the relationships between events: clocks timestamping events might not be related between them.
- *Cause*: a dependence relationship between activities. An activity (event) depends upon other activities (events) if it happened only because the other activities (events) happened. If event B depends upon event A, then A *caused* B. If neither caused the other, they are *independent*.[1]
- *Aggregation*: if event A signifies an activity that consists of the activities of a set of events $B_1, B_2, \ldots, B_n$, then A is an *aggregation* of all the events B_i. Conversely, B_i

[1]This computational notion of causality is ostensibly more limited than the notion of causality in philosophy and science in general: an interested reader is referred to [Pearl, 2009].

are *members* of A. Aggregation is an abstraction relationship: usually event A is created when a set of events $\{B_i\}$ happens. A is a higher-level event and we call it a *complex event*. A's members are the events that *caused* it. Aggregation can be referred to also as *vertical causality*.

In the context of this research, CEP aims at identifying such aggregation rules, so to make the activities in a complex system understandable to humans. Aggregation thus is a fundamental component in an *event abstraction hierarchy* that consists of: a *sequence of levels of activities*: each level consists of a set of descriptions of activities and, for each activity, a specification of the events that signify instances of that activity; and a *set of event aggregation rules for each level*.

Authors in [Cugola and Margara, 2012] introduce the field of *information flow processing*, which includes *data stream management systems* (DSMS) and CEP systems. While there are many similarities between DSMS and CEP approaches, there are also some key differences. DSMSs specialize in dealing with transient data that is continuously updated with an approach similar to database management and usually relying on SQL-like operators. This includes selections, aggregates, joins and similar operations.

Conversely, [Cugola and Margara, 2012] defines CEP models as viewing the flows of information as notifications that need to be filtered and aggregated into a higher-level understanding. In this sense, they describe CEP as an expansion of the *publish-subscribe* model that allows subscribers to express interest in the *complex events*, which are composed of multiple *simple events*. As such, the input stream, which is viewed as a stream of simple events, publishes each of the events that occur. However, users are not interested in those simple events directly. Instead, they subscribe to the complex events, which are aggregations of said simple events based on a set of rules. The task of the CEP engine is to aggregate those simple events into the appropriate complex events, which are then published to the users.

However, since the publication of [Cugola and Margara, 2012], the differences between DSMS and CEP approaches have become less substantial, as both types of approaches take inspiration from each other. In fact, this was already predicted and encouraged by the authors of [Cugola and Margara, 2012]. The term information flow processing, however, has become mostly disused in recent years. As such, while some of the approaches discussed in this chapter would arguably be closer to DSMS than CEP, we use terms from the CEP field. This is done to avoid the confusion that would arise from having different terms for very similar (if not the same) concepts.

To differentiate between approach types, we have decided to divide CEP into two groups: symbolic and non-symbolic-focused. Symbolic approaches allow users to define rules to identify when complex events are occurring. These approaches tend to provide the user with very flexible rule definitions, allowing the users to specify with a large degree of detail what are the conditions under which a complex event happens. However, they are generally limited in the types of input data that they can use. Since the user is required to define rules to process the data directly, non-symbolic data cannot be used as an input.

Our research has been centred on non-symbolic input data. As such, we will mainly look at non-symbolic-focused approaches, which are a newer area of research in the field of CEP. As the name implies, these are approaches that are designed to be able to deal with non-symbolic data. This allows them to make use of input streams containing types of data such as images, video, audio or text. However, most existing non-symbolic-focused approaches significantly limit the flexibility of rule definitions. Some even remove such a possibility completely: while beneficial in some cases, it does make it harder to detect complex events with more composite definitions. As such, our research has been focused on developing an approach that is able to deal with non-symbolic data while retaining the flexibility in rule definitions from the symbolic approaches.

In the following sections, we explore some of the approaches for both groups.

2.2 Symbolic CEP approaches

In symbolic approaches, users are expected to provide aggregation rules by specifying under which conditions a complex event is happening. These conditions are generally defined based on the simple events that have recently occurred. Some approaches also allow evaluating the state of other complex events to control whether a complex event occurs or not. For instance, some approaches may allow the user to define that complex event A to happen if complex event

B is happening at the time. An interested reader is referred to [Robins, 2010], which examines the foundations of symbolic CEP approaches.

The rules defining when complex events occur are generally defined by experts in the field for these approaches. As such, a person who understands the conditions under which complex events occur is required for this type of approach.

Symbolic approaches have been used in many different applications in a number of areas, including areas like business activity monitoring [Teymourian et al., 2012], sensor networks [Anicic et al., 2012b] and weather reports [Anicic et al., 2012a]. These approaches are ideal when dealing with symbolic data for which we have experts capable of correctly defining rules. However, in this thesis we focus on dealing with non-symbolic data, which these approaches are unable to handle effectively. This is because they require human experts to manually define the rules specifying the complex events based on the input data. While theoretically possible, this is usually unfeasible as the complexity of the rules would be far beyond human capabilities. For example, while someone could, in theory, define rules to identify which digit appears in a given image, the complexity and length of those rules make it infeasible to do so by hand. Doing the same to detect even more complex problems, such as identifying the type of sound from a section of a recording based on given rules, is impossible in practice.

2.2.1 Rule learning approaches

Some symbolic approaches also allow the system to learn the rules. These approaches can be used in any situation where the rules for the complex events are not known. For example, they can be useful if no experts are available, or when the aggregation rules are not fully known.

There are many different approaches to learning such rules. [Bruns et al., 2019] uses genetic programming to find the rules for several different scenarios. On the other hand, the approach in [Margara et al., 2014] aims to learn the rules by dividing the task between 7 modules, each tasked with learning a different part of the complex event rule. [Mehdiyev et al., 2015] compares 6 different rule-based machine learning approaches. However, these approaches all fall short of learning rules able to process non-symbolic data directly, as this would require very complex rules. Instead, these approaches are designed to generate rules on a similar level of complexity to the rules that can be generated by humans.

2.2.2 Prob-EC

An article of particular importance for the background of this thesis is [Skarlatidis et al., 2015], as our systems follow implementations inspired by the approach presented, Prob-EC. Prob-EC is a system for recognizing human activities from a symbolic representation of a video. In [Skarlatidis et al., 2015] authors show how the system can be used to detect complex events such as a person leaving an object, people meeting, moving together or fighting using the first dataset of the CAVIAR project[2]. In order to detect such complex events, the system takes into consideration five types of simple events (abrupt movement, walking, running, active and inactive) as well as the coordinates of tracked people and objects. The authors then manually define rules that aim to detect the complex events described above based on combinations of these simple events.

Prob-EC uses an approach that makes use of a dialect of Event Calculus to represent complex events. For this purpose, it defines complex events as fluents, which can be initiated and terminated. Then, it considers that complex events start happening when it is initialized, and end when they are terminated. Users can determine when complex events are initiated and terminated by specifying rules that define under which conditions this happens. These definitions are specified using ProbLog rules (for more details on ProbLog, see Section 2.2.2 just below).

When trying to determine during which period the complex event is occurring, multiple initializations may happen in a row. If we know for a fact that each of them have happened (that is, they have a probability of 100%), the complex event will start with the first initialization and last until it is terminated (with further initializations between those not causing any change). Similarly, a termination when the complex event is not happening does not change the state of it either. Instinctively, this functions in a similar manner to traffic signs. For example, if you pass a *no-overtaking* sign, this means that overtaking is forbidden until passing an *end of no-overtaking* sign. In this context, we could consider *overtaking forbidden* as the complex event. Then, passing a *no-overtaking* sign would initiate the complex event while passing an *end of no-overtaking* sign would terminate it. Finding further *no overtaking* signs when overtaking

[2]Available at `http://homepages.inf.ed.ac.uk/rbf/CAVIAR/` (on 20th of April 2022)

is already forbidden does not change the fact that you are not allowed to overtake. Similarly, although less common, passing an *end of no-overtaking* sign when overtaking is allowed does not change the fact that overtaking is permitted either.

This becomes a bit more complex when we are not entirely certain about the initializations and terminations. In those cases, initializations increase our confidence that the complex event is happening in an amount proportional to our confidence on the initialization. As such, multiple initializations in a row can increase our confidence that the complex event is occurring to higher levels than any individual one (up to 100% probability). By contrast, terminations decrease our confidence that the complex event is happening proportionally to our confidence in the termination. As such, consecutive terminations can decrease our confidence to effectively 0, even if we are not sure about them individually.

Something worth noting is that the Prob-EC approach does not directly interact with the video. Instead, Prob-EC uses the simple event labels, which have been manually assigned to each timestamp in each of the videos in the dataset. For instance, Listing 2.1 shows one of the rule definitions used in [Skarlatidis et al., 2015], which defines that the complex event *leaving an object* starts when there is an inactive object (lines 2 and 3) that is close to a person (lines 4 and 5). The complex event terminates if the object disappears (lines 7 and 8).

```
1 initiatedAt(leaving_object(Person, Object) = true, T):-
2         happensAt(inactive(Object), T),
3         happensAt(appear(Object), T),
4         holdsAt(close(Person, Object, 30) = true, T),
5         holdsAt(person(Person) = true, T).
6
7 initiatedAt(leaving_object(_, Object) = false, T):-
8         happensAt(disappear(Object), T).
```

Listing 2.1: Prob-EC rule definitions for the *leaving an object* complex event.

As shown in Listing 2.1, two types of clause are used to define when complex events are initialized and terminated in Prob-EC. The first type uses the clause `happensAt`, which, in this case, defines when people and objects appear and disappear in the video, as well as what actions they are taking (active, inactive, walking, running, ...). All these clauses come from a manual labelling of what is happening in the video. The second type of clause is `holdsAt`, which determines if other complex events are happening at the given time or not. In the example shown in Listing 2.1, these complex events include two entities being close to each other (determined based on a given distance) and one of the entities being a person. These complex events are, in turn, defined based on other `happensAt` clauses (which will also be manually labelled) and, possibly, other complex events. Of course, this means that Prob-EC can only be used in videos for which someone has manually labelled the simple events. As such, while Prob-EC is a good demonstration of the potential of using CEP on such types of data, it is not a practical solution for detecting such complex events from general videos. In Section 2.3 below, we discuss newer approaches that do attempt to deal with non-symbolic data. Furthermore, in the rest of the thesis, we explore how approaches like Prob-EC could be combined with neural networks to allow them to use non-symbolic data inputs in an end-to-end manner. That is, systems that are able to output predictions on when the complex events are happening using only the non-symbolic input data, without requiring any further input from humans or other systems.

In the following section, we explore ProbLog, the programming language used to define rules in Prob-EC. ProbLog has also been used in ProbCEP and DeepProbCEP, two of the approaches presented in this thesis.

ProbLog

ProbLog [De Raedt et al., 2007, Fierens et al., 2015] is a probabilistic logic programming language. ProbLog allows users to encode complex interactions between different components. A ProbLog program consists of a set of probabilistic facts F and a set of rules R. Facts have the form $p :: f$ where p is a value between 0 and 1, which represents the likelihood of the fact being true, and f is an atom. Atoms are expressions of the form $q(t_1, ..., t_n)$ where q is a predicate and t_i are terms. Rules have the form $h :- b_1, ..., b_n$ where h is an atom and b_i are literals. A literal is an atom or the negation of an atom.

One convenient syntactic extension is an annotated disjunction (AD), which is an expression of the form

$$p_1 :: h_1; ...; p_n :: h_n :- b_1, ..., b_m.$$

where the p_i are probabilities so that $\sum p_i = 1$, and h_i and b_j are atoms. The meaning of an AD is that whenever all b_j hold, h_i will be true with probability p_i, with all other h_i false (unless other parts of the program make them true). This is convenient to model choices between different categorical variables. ProbLog programs with annotated disjunctions can be transformed into equivalent ProbLog programs without annotated disjunctions [De Raedt and Kimmig, 2015].

```
1 0.05 :: burglary.
2 0.01 :: earthquake.
3 0.45 :: at_home(john).
4 0.65 :: at_home(mary).
5 alarm :- burglary.
6 alarm :- earthquake.
7 calls(X) :- at_home(X), alarm.
```

Listing 2.2: Simple example of a ProbLog program.

Listing 2.2 shows a simple example of a program in ProbLog. Firstly, the program defines the probabilities of a burglary and an earthquake happening where our house is. Then, lines 3 and 4 define how likely it is for our neighbours, John and Mary, to be home. All these probabilities should be based on previous experience. Then, lines 5 and 6 define that our alarm will be activated if there is a burglary or an earthquake, respectively. Finally, line 7 defines that our neighbours will call us if they are home and they hear the alarm. In order to define the rule for all our neighbours, the variable X is used. This would then allow us to perform queries on the program. For instance, we could perform the query $calls(mary)$. This would return us the probability of Mary calling us.

ProbLog considers the ground probabilistic facts as independent random variables, i.e., we obtain the following probability distribution P_F over truth value assignments to sets of ground facts $F' \subseteq F$:

$$P_F(F') = \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i)$$

As each logic program obtained by choosing a truth value for every probabilistic fact has a unique least Herbrand model, P_F can be used to define the *success probability* $P(q)$ of a query q, that is, the probability that q is true in a randomly chosen program, as the sum over all programs that entail q:

$$P(q) := \sum_{\substack{F' \subseteq F \\ \exists \theta F' \cup R \models q\theta}} P_F(F') = \sum_{\substack{F' \subseteq F \\ \exists \theta F' \cup R \models q\theta}} \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i)$$

ProbLog Inference After a ProbLog program has been created, it can be used to infer the answer for queries. This process will tell the user what is the probability that the queried element is true, given the definition of the program and the evidence provided by the user.

Inference in ProbLog is concerned with computing marginal probabilities of queries, i.e., ground atoms, under this distribution, potentially conditioned on a conjunction of evidence atoms. While this is a #P-hard problem in general, ProbLog relies on state-of-the-art knowledge compilation techniques to achieve scalable inference across a wide range of models.

The inference process requires a ProbLog program L, evidence $E = e$ and a set of query atoms Q. In the first step, L is grounded based on Q and $E = e$, resulting into L_g. This removes any part of the program that is irrelevant for the given Q and $E = e$. That is, only the parts of the program that are needed to calculate $P(Q|E = e)$ are present in L_g.

Then, the rules in L_g are converted to an equivalent Boolean formula φ_r. This could be used to perform weighted model counting directly. However, this is not efficient. In order to perform the calculation more efficiently, φ_r (combined with the evidence) is transformed into an *Arithmetic Circuit (AC)*. Finally, this AC can be used to calculate the result for the queries.

Figure 2.2 shows the AC resulting from performing the query $calls(mary)$ on the code shown in Listing 2.2 above. We can see that the probability that Mary calls depends on the probability that she is home, combined with the probability that the alarm is activated, which in turn depends on the likelihoods of an earthquake and burglary happening. We can also see that John is not present in this arithmetic circuit. This is because the probability of John calling or being home has no effect on the probability of Mary calling, according to the rules we have defined.

For a more detailed explanation of the inference process, see [Fierens et al., 2015].

2.3 Non-symbolic-focused CEP approaches

In the previous section we explore the symbolic CEP approaches, which have historically been where the CEP field has focused on. However, as argued in [Yadav, 2019], while these approaches are well suited for dealing with symbolic data, they are not able to deal with video inputs. According to [Yadav, 2019], this is due to four reasons: (i) it is challenging to write

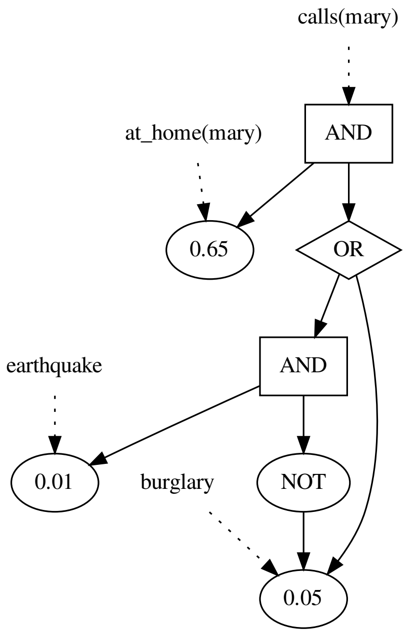


Figure 2.2: Arithmetic Circuit for the burglary case defined in Listing 2.2.

rules to detect what is happening in a video based on the video content (pixel values), (ii) the low-level features (pixel values) do not match the high-level understanding humans are used to (for example, *person*), (iii) dealing with video introduces object and spatial components, which make pattern matching harder (for example, considering how a person is moving through a video is much more complex to evaluate than tracking the change in the temperature detected by a sensor) and (iv) video data is more computationally intensive than most symbolic data. We would go further in this assessment and argue that these arguments apply to all types of non-symbolic data. As such, we would claim that symbolic CEP approaches are not suited for dealing with non-symbolic data in general, thus including, for example, audio and images. Based on our definition, non-symbolic data includes any type of data for which rules cannot (easily) be manually defined to extract information. This is because humans cannot intuitively understand the low-level features of non-symbolic data (for example, pixel values), as they are used to thinking in high-level values (for example, people or objects).

As such, in order to deal with non-symbolic data we need another type of approach. For the purpose of this thesis, we refer to such approaches as *non-symbolic-focused* CEP approaches. While some of these approaches are still able to deal with symbolic data, they are all designed with the intent of being able to deal with symbolic data. There are three main types of non-symbolic-focused approaches: (i) neural-only approaches, (ii) pre-trained neural network approaches and (iii) neuro-symbolic approaches. In the following sections, we explore the advantages and disadvantages of each type of approach.

2.3.1 Neural-only approaches

One approach when dealing with non-symbolic data is to treat the whole CEP problem as a classification problem. This allows for the use of machine learning approaches such as neural networks. We generally refer to this type of approach as *neural-only*.

These approaches opt to remove the manual definitions of the complex events and instead attempt to train the neural network to identify those definitions at the same time as it learns to classify the non-symbolic data.

Due to the relevance of time in the definition of complex events, a Long Short Term Memory (LSTM) [Mishra et al., 2018] or a Convolutional 3D (C3D) [Liu et al., 2018] can be used. Thanks to the fact that these neural networks are trained in an end-to-end manner without requiring any input in the form of manually defined rules, these approaches can be used even when experts are not able to define the conditions under which a complex event occurs.

However, this advantage comes at the significant cost of requiring large amounts of training data. This becomes even more relevant when the complexity of the rules increases, as more training data is required to appropriately train the system for all the cases. As such, while these approaches can work well if the rules for the complex events are fairly simple and there is a lot of training data, they tend to fall behind in cases where training data is more sparse.

2.3.2 Pre-trained neural network approaches

Other approaches choose to instead make use of a pre-trained neural network, which is used to extract the symbolic information from the non-symbolic data. This extraction allows users to define symbolic rules on the data, which can be used to define the rules to detect complex events. For this extraction process, the non-symbolic data is usually split into atomic pieces of information, which are treated as simple events.

This approach has been used in [Roldán et al., 2020], where the authors show that it makes it possible to reduce the number of false positives when detecting IoT security attacks. They use a neural network to predict the length of the suspected packets. If the predicted length does not match the actual length of the packet, a complex event is generated indicating that an attack might be happening.

[Yadav, 2019] and [Yadav et al., 2021] also use approaches that rely on pre-trained neural networks to detect cases where cars illegally overtake from video and occupational health and safety compliance based on multimodal sources, respectively. In both cases, the input stream (or streams) are processed by an event knowledge graph, which is generated based on the query performed by the user. This graph defines how the input stream(s) need to be processed by the different available pre-trained neural networks, and how those outputs need to be combined to answer the query performed by the user, based on how they have defined the complex events. For example, a user can define a query that checks the occupational and health safety compliance by checking that the noise levels are below a certain threshold (evaluated based on the sound input stream) and that all workers are wearing a hat (evaluated based on the output of an object detector from the input video stream).

The main advantage of this approach is the fact that it does not require any training data. As such, it can easily be deployed in new settings, provided that the developers have access to a pre-trained neural network suitable for that purpose. As such, this type of approach can be quite agile and efficient. However, these approaches also have a significant limitation in the fact that, if the required neural network does not exist, it is not possible to use them. Of course, it might be possible to train such a neural network. However, training data and labels would be required for this, which can be difficult to obtain as the data would need to be labelled in accordance with the symbolic rules, instead of just indicating when a complex event is occurring.

2.3.3 Neuro-symbolic approaches

Neuro-symbolic approaches function in a similar manner to pre-trained neural network approaches, as they also combine neural networks with symbolic rules. However, instead of using pre-trained neural networks, these approaches aim to train the neural network as part of the system in an end-to-end manner. This means that, as in the neural-only approaches described above, the only training data required is the input to the system and labels indicating when a complex event happens. This makes it easier to obtain training data than in the case of the pre-trained neural network approaches, as it is much easier to identify when a complex event has happened than to label every simple event that happens in a given stream. As such, they have been described as quite promising in the field of event processing [Khan and Curry, 2020]. Furthermore, when compared with neural-only approaches, the main advantage of this type of approach is the fact that they are very *training data-efficient* [Susskind et al., 2021], making them a good choice for situations in which it is difficult to obtain enough data to train other approaches. [Apriceno et al., 2021] has also shown that CEP neuro-symbolic approaches are able to generalize to unseen outcomes, which can be useful regarding Challenge 5 from Chapter 1. This is thanks to the fact that neuro-symbolic approaches learn to classify the simple events, which are then combined based on the symbolic rules. As such, complex events unseen in the training data can be correctly identified, as long as the system has learnt how to classify the relevant simple events correctly (which can be learnt based on other complex events) and the rules for the unseen complex event are correctly defined.

Neuro-symbolic approaches have also been shown to improve the quality of AI generated captions for a video. While not being exactly the same problem as CEP, many points are still similar, as the system still needs to detect what is happening based on the input video in order to generate the captions. Of course, this is not exactly the same as detecting when a pre-defined complex event is occurring, but it still requires the system to understand what is happening in the video. In [Akbari et al., 2020], authors demonstrate that the use of a neuro-symbolic approach can improve the quality of the generated captions by learning the relations between videos and their paired text descriptions. More specifically, their proposed system learns a set of discrete roles, meaning that the model can train for different domains while still being able to disentangle the actions for each of those domains.

Another field that has similarities with CEP and has shown potential for neuro-symbolic approaches is context understanding [Oltramari et al., 2020]. Context understanding is an agent's ability to consider multiple sources of information for decision-making. For instance, an automatic driving car needs to understand the position of cars and pedestrians around it, as well as what each of those cars and pedestrians is likely to do, in order to take a decision. While this is not looking for a specific complex event, it still requires an understanding of the agent's environment. In [Oltramari et al., 2020], authors show that neuro-symbolic approaches have potential in the field of context understanding, by testing neuro-symbolic models in two applications: (i) automated driving and (ii) question answering.

However, while multiple neuro-symbolic approaches do exist in the area of AI in general, there are not many in the area of CEP specifically, at the time of writing. As detailed in the following section, which explores the types of neuro-symbolic approaches present in AI in general, there are different types of neuro-symbolic approaches. However, they all combine neural networks and symbolic rules with the aim of capturing the strengths of both fields. While this provides various advantages, it also tends to introduce some limitations. For example, in most cases the types of neural networks that can be trained using a neuro-symbolic approach are limited. However, there are also ways to work around this problem. For example, authors in [Apriceno et al., 2021] propose a CEP approach that makes use of a pre-trained object detector to detect the positions of MNIST digits in a given frame from a video. Then, they train an object classifier to identify the value of the detected MNIST digits. This is used to detect different complex events from a video feed. These complex events consist of two MNIST digits moving towards each other during the duration of the video, and combining by either adding together or subtracting from each other. In this case, a pre-trained object detector is used as it would be difficult to train it in an end-to-end manner with the approach used by the paper. However, using this pre-trained object detector allows the authors to train the object classifier in an end-to-end manner.

Another disadvantage of the neuro-symbolic approaches is that they have less potential for parallelism than neural-only models, due to their symbolic component [Susskind et al., 2021]. This can make neuro-symbolic approaches slower to train, as this makes it harder to perform batch training. As such, neuro-symbolic approaches will usually require more time to train.

In the following section, we explore the different types of neuro-symbolic approaches that can be used, as well as discussing the advantages and disadvantages that each of them may have for the problem we are trying to solve.

2.4 Types of Neuro-Symbolic Approaches

According to [Besold et al., 2017], neuro-symbolic approaches are those that provide either (i) a neural implementation of a logic, (ii) a logical characterisation of a neural system, or (iii) a hybrid learning system that brings together features from neural and symbolic approaches. In this thesis, we focus on the third group where neural networks are combined with symbolic approaches with the aim of capturing the strengths of both approaches. Generally, this divides the approaches into two layers, a neural layer that can be trained to extract information from non-symbolic data, which performs the task of low-level perception, and a symbolic layer where rules can be defined, which performs the high-level reasoning.

Using a neural layer to extract information from the input data allows this approach to circumvent one of the main issues with symbolic approaches; namely, converting the raw data into an abstract form that can be processed by the symbolic approach. If done manually, this often implies drastic information loss, and thus the predictions given by the system no longer accurately represent the real world. By contrast, neural networks can be used to replace this manual extraction, resulting in significant gains in performance.

On the other hand, it has been demonstrated that the inclusion of a symbolic layer can significantly reduce the amount of training data required. In [Susskind et al., 2021], authors compare a neuro-symbolic approach —Neuro-symbolic Concept Learner [Mao et al., 2019]— with two pure deep learning models —TbD [Mascharka et al., 2018] and MAC (Memory, Attention and Composition) [Hudson and Manning, 2018]— after training with both 100% and 10% of the training data from the CLEVR dataset [Johnson et al., 2016], which tests a range of visual reasoning abilities. The authors show that all three approaches obtain similar accuracies of about 99% when trained using all the training data. However, the pure deep learning models (TbD and MAC) perform significantly worse when using only 10% of the training data, with accuracy results of 54.2% and 67.3% respectively. In contrast, the neuro-symbolic approach retains an accuracy of 98.9%, meaning that the performance only drops by 0.3% despite only using 1/10 of the training data.

It is for this reason that we believe neuro-symbolic approaches have a great potential in the research area of complex event processing using non-symbolic data, as the nature of complex events lends itself well to being defined using a symbolic approach. Furthermore, the capability of neuro-symbolic approaches to obtain good performances in situations where the training data is sparse makes them an ideal choice for CEP problems where it can be hard to find enough training data for more traditional machine learning approaches.

In this section, we explore the different types of neuro-symbolic approaches. In particular, we explain the benefits of the approaches used in the following chapters of the thesis regarding the problems we are trying to solve.

Following [De Raedt et al., 2019], approaches that combine neural networks with logic can be divided into two categories: (i) logic as constraints and (ii) differentiable logic frameworks.

2.4.1 Logic as constraints

This group of approaches use the logic rules as a constraint on the neural network. This is done by using a regularization term derived from the logical properties. This regularization term encourages the neural network to obey the logical reasoning by adding a penalty to the loss function if the logic rules are not followed. The specific manner in which the rules can be specified differs between approaches. For example, the approaches proposed in [Diligenti et al., 2017] and [Donadello et al., 2017] use first-order logic, while other approaches use *if-then* rules [Demeester et al., 2016, Minervini et al., 2017].

Despite these differences, all these approaches encode the logic into the parameters through the use of regularization. This means that the constraints are soft; there is no guarantee that they all will be satisfied. Instead, there will be a trade-off between enforcing the constraints and following the data if these are contradictory, which can be controlled by a hyperparameter of the model. As such, the systems generated with these types of approaches may give results that are not consistent with the logic they were provided [Xu et al., 2018].

This could certainly be beneficial in CEP situations where the rules for complex events are not fully known. For example, this could allow experts to provide a rough definition of the rules as they understand them, which would help the system by reducing the amount of training data required. However, the system would also be able to learn to identify the cases in which the rules are incorrect by learning from the training data.

For our research, however, we are assuming that experts do know the exact rules for the complex events. Furthermore, we want our approach to be *reliable*, meaning that we want to avoid using approaches that may give outputs that are not consistent with the rules we have defined.

2.4.2 Differentiable logic frameworks

The other group of approaches to neuro-symbolic architectures aims to make the logic layer differentiable, which allows the system to backpropagate through the logic program and thus train the neural network. In [De Raedt et al., 2019], authors say that this can be achieved through the use of the mathematics of differentiable functions, which can be used to reformulate the basic reasoning primitives. That is, the original logic rules defined by the users are transformed into a differentiable function, usually a neural network. Examples of this approach include [Rocktäschel and Riedel, 2017], [Evans and Grefenstette, 2018] and [Cohen et al., 2018].

However, these approaches have restrictions in the classes of logic programs they allow. As defined above in Chapter 1, we want our approach to retain flexibility in rule definitions. As such, these approaches do not seem ideal for our implementation either.

Knowledge Distillation

Another type of approach we would argue belongs to this group is the use of knowledge distillation [Hinton et al., 2015, Hu et al., 2016]. These approaches use student-teacher techniques where the student learns to replicate the behaviour of the teacher. They have been shown to have a large number of applications, generally focusing on program acceleration. This is archived by using a neural network to emulate a slow part of the problem, which increases speed at the cost of accuracy.

However, the same approach can be used to generate a student neural network that emulates a symbolic program, thus allowing for a neuro-symbolic approach. This is made possible by combining the student neural network, which implements the symbolic part, with another neural network. As both parts are implemented using neural networks, the system is differentiable, allowing for end-to-end training. However, as the student neural network emulates the functionality of a symbolic program, this provides the advantages of both neural and symbolic approaches.

In order to archive this, the knowledge distillation approaches generate large amounts of data, which is automatically labelled using the teacher system. In the context of neuro-symbolic approaches, the teacher will be a symbolic program, meaning that the student model will result in a neural network that behaves as the symbolic program. Using this generated data, combined with the labels provided by the teacher system, a neural network is trained to emulate the symbolic program. The resulting student neural network can then be used as high-level reasoning and combined with another neural network, which can perform the task of low-level perception. The weights for the student neural network can then be frozen, meaning that they will not change with further training. As such, when performing end-to-end training, the part of the system responsible for low-level perception will be trained to identify the inputs required by the high-level reasoning part of the system.

The knowledge distillation approach has some drawbacks, however. Firstly, the user is required to define the architecture for the student neural network. As such, the user needs to find the appropriate neural network architecture for representing the symbolic rules they are using. Furthermore, changes in the rules may also require this architecture to change.

Another limitation is the fact that, similarly to the approaches that use logic as constraints, this approach does not guarantee that all the constraints will be satisfied. While the constraints will generally be followed more strictly by this approach, the nature of neural networks means that there is no guarantee that the output will be consistent with the logic that has trained it. Training the student with larger amounts of data may reduce this risk, but it is not possible to entirely eliminate it. This is in a similar manner to how neural networks, even when trained to high accuracy, will still miss-classify some of the testing instances. While this is unlikely to happen when using simple rules, more complex rules will increase the likelihood, while also making it more difficult to detect due to the high number of possible combinations that would need to be evaluated to ensure this does not happen.

DeepProbLog

Finally, DeepProbLog [Manhaeve et al., 2018, Manhaeve et al., 2021] is another type of neuro-symbolic approach which we would argue fits in the group of differentiable logic frameworks. However, instead of transforming the logic rules into a neural network as the previous approaches, DeepProbLog extends the probabilistic logic programming language ProbLog [De Raedt et al., 2007] (explained above in Section 2.2.2) with neural predicates. Neural predicates can be used to define facts where the likelihood of being true is given by the output of a neural network. This allows DeepProbLog to consult a neural network as part of the logic program. Furthermore, the fact that the logic layer is probabilistic also makes it differentiable, which is why we would argue that it belongs in this group of approaches. This means that any program written in ProbLog is differentiable, even if it does not contain any probabilistic facts. As such, DeepProbLog is very *agile* as it can provide a lot of flexibility in how the rules can be defined. This is thanks to the fact that ProbLog, as an extension of Prolog, is Turing-complete.

A DeepProbLog program is a ProbLog program that is extended with a set of ground neural ADs (nADs) of the form $nn(m_q, [X_1, ..., X_k], O, [y_1, ..., y_n]) :: q(X_1, ..., X_k, O)$. Here, nn indicates that the following is an nAD and m_q is a neural network identifier. The neural network m_q will be provided the input vector $[X_1, ..., X_k]$ and output a probability distribution over the domain $O \in [y_1, ..., y_n]$. nADs work similarly to ADs in the sense that they provide a mutually-exclusive distribution of probabilities over a set of clauses. In nADs, however, these probabilities are generated from the output of a neural network, instead of being manually defined. For instance, if we wanted to classify MNIST digits into their respective values so that we can perform operations with them, we would specify the following nAD:

```
nn(mnist_net,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).
```

This would then allow us to perform a query of the form `digit(` $, Y)$. This query would be transformed into an AD of the form

p_0 :: `digit(` $, 0); p_1$ :: `digit(` $, 1); ...; p_9$ :: `digit(` $, 9)$

The values of the probabilities p_i would be based on the output of the neural network. This neural network could take any shape, e.g., a convolutional network for image encoding, a recurrent network for sequence encoding, etc. However, its output layer, which feeds the corresponding neural predicate, needs to be normalized. In neural networks for multiclass classification, this is typically done by applying a softmax layer to real-valued output scores.

After defining the structure of the neural network and the logic layer, it is possible to use DeepProbLog to infer the answers to our queries. In order to perform this inference, DeepProbLog performs several steps that transform the logic layer into an arithmetic circuit. These steps are mostly the same as in ProbLog (see Section 2.2.2 above), with the only difference that the nADs need to be instantiated according to the output from the neural network. This is done after grounding, and is archived by assigning the outputs of the neural network to the nAD, which transforms it into a regular AD. The rest of the inference steps for ProbLog are then followed, resulting in an arithmetic circuit that can be used to calculate the probability that the query is true [Manhaeve et al., 2021]. These steps are specific to each query. As such, the inference steps must be followed for each new query. It is possible to save the arithmetic circuit for a given query if that query needs to be performed multiple times (such as in the case of training over multiple epochs). However, the amount of memory required to have a cache of arithmetic circuits for large numbers of queries makes this approach infeasible for larger datasets.

In order to train the neural network, the system first performs inference as described above. Then, DeepProbLog is able to perform gradient-based learning. First, the arithmetic circuit used during the inference is also used to perform the gradient computations. Since this arithmetic circuit is composed of addition and multiplication operations, this means that it is differentiable. This allows DeepProbLog to compute the gradient with respect to the probabilistic logic program. This gradient can then be used to train the neural network using backpropagation. For a more detailed explanation of the technical aspects of DeepProblog's inference and learning, see [Manhaeve et al., 2021].

Focus	Type of approach	Examples	Allowed data (symbolic/ non-symbolic)	Rule definitions	Requires training
Symbolic	SQL-like approaches	[Bezerra et al., 2021], [Burgueño et al., 2018]	Symbolic	SQL-like languages	No
	Specific CEP languages	[Chapnik et al., 2021], [Yankovitch et al., 2022]	Symbolic	Specific CEP language	No
	Logic	[Skarlatidis et al., 2015]	Symbolic	Logic rules in ProbLog	No
	Rule-learning	[Bruns et al., 2019], [Margara et al., 2014], [Mehdiyev et al., 2015]	Symbolic	Rules are learnt (various approaches)	Yes (rule learning)
Non-symbolic	Neural-only	[Mishra et al., 2018], [Liu et al., 2018]	Non-symbolic	No rule definitions	Yes
	Pre-trained neural networks	[Roldán et al., 2020], ProbCEP	Both	Graphical tool, ProbLog rules	No training (pre-trained neural networks required)
	Neuro-symbolic through knowledge distillation	Neuroplex	Non-symbolic	Specific CEP language (currently only sequences of simple events supported)	Yes (reasoning and perception layers)
	Neuro-symbolic using DeepProbLog	[Apriceno et al., 2021], DeepProbCEP	Both	Logic rules in ProbLog	Yes (perception layer)

Table 2.1: Summary of the CEP approaches discussed in this chapter and later in the thesis. The table is divided between symbolic and non-symbolic-focused approaches. For each type of approach, examples of models are provided. We also detail whether the allowed types of input data include symbolic, non-symbolic or both (*Allowed data* column), as well as how the rules are defined for each approach (*Rule definitions* column). Finally, we detail whether training is required for each approach.

2.5 Conclusion

Table 2.1 shows a summary of the CEP approaches discussed in this chapter, as well as those in which we have worked as part of this thesis, which are explored in more detail in the rest of the thesis. The first half of the table shows the symbolic approaches. For all of these, while the specific manner in which rules are defined does vary, they are all consistent in the fact that they only allow symbolic data as an input, and that most of them do not require training. This is the area where most CEP work has focused historically, making those approaches very appropriate to perform CEP on symbolic data. However, their inability to deal with non-symbolic data prevents them from using sources such as images, audio or video, among many others. The only outliers in the symbolic side are rule-learning approaches, which are capable of learning the rules from training data, which removes the requirement of manually defined rules. However, these approaches are still not able to deal with non-symbolic data.

On the other hand we have the non-symbolic-focused approaches, which are designed to be able to deal with non-symbolic data. As shown in Table 2.1, if the approaches proposed in this thesis are excluded, this is a fairly unexplored area of research, which is why the research presented in this thesis aims to fill this gap. There are some neural-only approaches in this area, which have the added benefit of not requiring rule definitions. However, as we demonstrate later in the thesis, these neural-only approaches are not appropriate in situations where there is little to no access to training data, which is fairly likely to happen due to the fact that complex events are usually relatively uncommon, as they tend to represent unusual situations. As a result, our focus is in situations where the input data is non-symbolic but there is sparse (or even no) training data. Since this PhD was started, some approaches have appeared that also focus on the same area. For instance, [Roldán et al., 2020] uses a similar approach to ProbCEP where pre-trained neural networks are used to deal with the non-symbolic data. However, [Roldán et al., 2020] focuses more on providing a user-friendly graphical interface that allows users to easily define and understand the complex events, whereas ProbCEP focuses on the use of proxy models to detect events for which we have no training data. On the other hand, [Apriceno et al., 2021] proposes a similar approach to DeepProbCEP, but the focuses of each research have been significantly different. The focus of [Apriceno et al., 2021] was centered around predicting complex events that were unseen in the training data, as well as the combination with a pre-trained object detector to allow the system to identify the position of elements in a frame of a video. Instead, our research using DeepProbCEP was more focused on the comparison of various approaches when training data is sparse, as well as an evaluation of the agility and robustness of the approach.

In this area of research, only using symbolic approaches is unfeasible due to the non-symbolic nature of the inputs, and using neural-only approaches is ineffective, as they require too much training data. As such, the approaches proposed in this thesis focus on combining neural and symbolic approaches, in order to make use of the benefits offered by both types of approach.

Some new CEP approaches do allow the use of non-symbolic data. However, these approaches tend to significantly limit the flexibility of the rule definitions, or even remove it entirely. While this can be useful in certain scenarios, it also prevents the users from being able to define complex events as specifically as they may want to. This does not only usually increase the amount of training data required by the system by a fair margin, but it also may cause the system to detect the complex events incorrectly. Our proposed approaches focus on allowing the user to define the rules for the complex events with as much flexibility as possible.

Chapter 3

Using Pre-Trained Neural Networks as Proxy Models For CEP

In this chapter, we introduce ProbCEP, an approach that makes use of proxy models combined with complex event definitions to perform CEP on non-symbolic data. Figure 3.1 shows a general schematic for the ProbCEP approach. ProbCEP allows the user to define under which conditions the complex events are generated (KR in Figure 3.1). These definitions are based on the outputs of proxy models, which consist of pre-trained neural networks (NN in Figure 3.1), albeit not with data about the type of complex events ProbCEP is trying to detect. As such, the ProbCEP approach is not designed to give a level of accuracy that competes with the state-of-the-art on detection for the problems it tries to solve. Instead, it is meant as an approach that does not require users to train a system to identify the situations they are interested in. This means that ProbCEP is intended for cases where a system needs to be deployed quickly and without having access to training data.

In this chapter, we describe how the ProbCEP approach could be used to detect violence from CCTV feeds. More specifically, we are trying to detect the abnormalities in the UCF-Crime dataset [Sultani et al., 2018] (see Section 3.4 below). For this purpose, we use an activity detec-

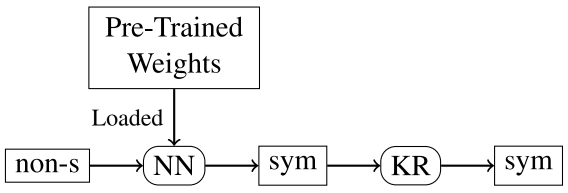


Figure 3.1: General schematic for ProbCEP. Rectangular boxes represent data, using *sym* for symbolic data and *non-s* for non-symbolic data. Oval boxes represent algorithms, using *NN* for neural network components and *KR* for knowledge reasoning (symbolic) components.

tion and an object detection pre-trained neural networks. However, neither of these are trained using a dataset that includes violence. Instead, the activity detection has been trained using a dataset with various human-focused actions (the Kinetics-400 dataset), while the object detection was trained with everyday scenes containing common objects in their natural context (the COCO dataset).

The use of proxy models in the ProbCEP approach is in contrast with traditional machine learning approaches, which tend to require large amounts of training data. This is because most machine learning approaches are not able to extrapolate knowledge about a test sample that is far different from the training data [Kaplan et al., 2018], meaning that the systems require training data about all possible situations in which the class that is being detected happens. As such, detecting classes that may occur in a large number of situations requires large amounts of training data. Violence is one of the cases in which this is particularly true, as it comprises a large spectrum of activities that can go from abuse, to fighting, to road accidents. Furthermore, these activities can take place in completely different environments, from public buildings, to underground stations, to roads during the day or the night. This is therefore one of those tasks at which humans excel, while machines still lag behind.

Moreover, situations in which violence occurs are relatively rare: this also makes it harder to obtain large amounts of training data to construct systems that are able to identify it automatically. The ProbCEP approach is designed to allow the detection of complex events despite the lack of training data. For this purpose, ProbCEP is designed to be *agile*, allowing for the use any type of input as long as the correct proxy model is used to process it. Particularly, ProbCEP is trying to fill the gap where the user wants to detect a situation from which not enough training data is available to use traditional machine learning approaches, but the type of input data does not allow for the use of purely symbolic approaches.

After showing that the ProbCEP approach is capable of detecting complex events using proxy models, we develop the PreCompilation approach, which allows the system to be more *time-efficient*, increasing the speed of the system and making it possible to keep up with live data. This is archived through the re-use of arithmetic circuits, which otherwise need to be compiled each time the system performs a query. As a reminder, an Arithmetic Circuit (AC) is the result of compiling the ProbLog program (see Section 2.2.2 above), which allows for a quick calculation of the result of the inference. As this compilation is the most time-consuming part of the system, this can significantly increase the speed of the system. The PreComplilation approach has been designed in a manner that makes it possible to use it in any ProbLog code that needs to perform queries with similar ACs a large number of times. More specifically, in regards to the ProbCEP approach, we show that PreCompilation allows the system to perform queries over 20 times faster.

Finally, we evaluate whether the ProbCEP approach could be used to identify the type of abnormal activity happening in the CCTV feed using the described proxy models. Our findings show that manually defining rules to differentiate the type of situation that is occurring could be challenging, which motivates our research for the rest of the thesis to be based on efficiently using small amounts of training data to learn to differentiate such situations.

3.1 Problem Definition

This problem definition is inspired by the Prob-EC approach [Skarlatidis et al., 2015]. As explained in Section 2.2, Prob-EC defines a system for recognizing complex events given a symbolic representation. This approach makes use of a dialect of Event Calculus to detect complex events. The main difference in our approach, however, is the fact that we support non-symbolic data as an input, unlike Prob-EC which only supports symbolic data.

Let $\mathcal{L}$ be a first-order language with three sorts, $\mathbb{S}$, $\mathbb{C}$ and $\mathbb{T}$, where $\mathbb{S}$ denotes simple events, $\mathbb{C}$ denotes complex events and $\mathbb{T}$ denotes timestamps. Timestamps consist of positive integers and zero, and are used to indicate at what time complex and simple events happen.

Simple events are assumed to have no time duration. As such, they can be mapped to a single timestamp with a function of sort $\mathbb{S} \rightarrow \mathbb{T}$, which indicates at what timestamp a given simple event happens. Similarly to Prob-EC, we can define that simple events happen at a certain time using clauses of the form $happensAt(\mathbb{S}, \mathbb{T})$. In this chapter, we will be using short segments of video as simple events. However, as their duration is so short we can assume that it happens in a single timestamp. Simple events can also have arguments, which may be used to consider whether complex events are occurring. Each type of simple event should have a pre-defined set of arguments, and each instance of that type should consider a value for that type.

On the other hand, complex events happen for a duration of time. We define complex events in a manner inspired by event calculus, where complex events are represented by *fluents*. These fluents have a value that can be evaluated at any point in time. We can express this using the predicate $holdsAt(\mathbb{C} = V, \mathbb{T})$, which indicates that the fluent $\mathbb{C}$ —and thus the corresponding complex event— has the value V at timestamp $\mathbb{T}$. This can also be expressed saying that the value V holds for fluent $\mathbb{C}$ at that point in time. Fluents maintain their value through time until they are explicitly changed, which can be done using the predicate $initiatedAt(\mathbb{C} = V, \mathbb{T})$. In order to define our complex events, we use $true$ and $false$ as the possible values that a fluent can take, where $true$ indicates that the complex event is occurring and $false$ indicates that it is not. As such, we consider that a complex event is being initiated if the value is being set to $true$ through the predicate $initiatedAt(\mathbb{C} = true, \mathbb{T})$. On the other hand, when we say that a complex event is being terminated we mean that its value is being set to $false$, using the predicate $initiatedAt(\mathbb{C} = false, \mathbb{T})$. With this, we can use the predicate $holdsAt(\mathbb{C} = true, \mathbb{T})$ to evaluate whether a complex event is occurring at a certain timestamp. For example, $holdsAt(violence = true, t)$ states that the complex event $violence$ is occurring at t. In order to determine which value holds for a fluent at any point in time, based on previous initializations and terminations, the following rules are used:

$$\begin{array}{ll} holdsAt(F = V, T) \leftarrow & broken(F = V_1, T_i, T_f) \leftarrow \\ \quad initiatedAt(F = V, T_s), & \quad initiatedAt(F = V_2, T_m), \\ \quad T_s < T, & \quad V_1 \neq V_2 \\ \quad \neg broken(F = V, T_s, T) & \quad T_i < T_m < T_f \end{array} \tag{3.1}$$

According to these rules, $holdsAt(\mathbb{C} = V, \mathbb{T})$ will be true if it has been initiated with that value and not $broken(\mathbb{C} = V, \mathbb{T}, \mathbb{T})$. A given value will be $broken(\mathbb{C} = V, t_i, t_f)$ if it has been $initiatedAt(\mathbb{C} = V_2, t_{m2})$, where $V \neq V_2$ and $t_i < t_{m2} < t_f$. That is, if the complex event is initialized with a different value $\mathbb{C} = V_2$, this will also *break* the complex event.

Since, as explained above, the approaches in this thesis only use $true$ and $false$ as possible values, users can define what conditions constitute an $initiatedAt(\mathbb{C} = true, \mathbb{T})$ predicate to happen. This determines under which conditions the corresponding complex event is initiated—that is, when that complex event starts. Similarly, users can define under which conditions

$initiatedAt(\mathbb{C} = false, \mathbb{T})$ happens, indicating when each complex event is terminated—that is, when each complex event ends.

Example 1. For instance, assume that we are trying to detect two people fighting in a video coming from a CCTV feed. In this case, the frame number can be used as a timestamp. Then, we can define under which conditions we believe two people are fighting, based on which types of simple events we are receiving. For this example, we will assume we are receiving the following types of events:

$$\mathbb{S} = \{punch, present(K), inProximity(K_1, K_2)\} \tag{3.2}$$

In this context, the simple event $punch$ indicates that a punch has happened. $present(K)$ can define that the object with identifier K appears in the video at the moment, where K is an argument for that simple event. Finally, $inProximity(K_1, K_2)$ indicates that the objects identified by K_1 and K_2 are physically close to each other at the moment.

In the rest of the thesis, we explore how the identification of such simple events from a non-symbolic input, such as a CCTV feed, can be archived through the use of neural networks. For this example, however, let us assume that we have some system capable of accurately identifying when such simple events happen.

Suppose also that we can use clauses of the form $type(K)$, which will be true if the object identified by K is of the $type$ defined by the clause. For instance, $person(K)$ would be true if K identifies a person. Again, this would likely be implemented using a neural network.

Then, we could define a fight to start when multiple punches happen in a short period of time. Furthermore, we know that at least 2 people need to be present for a fight to happen, and that those people need to be in close proximity. We can define these restrictions as follows:

$$
\begin{aligned}
&initatedAt(fight = true, t_i) \leftrightarrow \\
&\quad happensAt(punch, t_i), \\
&\quad happensAt(punch, t_{i+1}), \\
&\quad happensAt(present(P_1), t_i), \\
&\quad happensAt(present(P_2), t_i), \\
&\quad person(P_1), person(P_2), P_1 \neq P_2, \\
&\quad happensAt(inProximity(P_1, P_2), t_i)
\end{aligned}
\tag{3.3}
$$

This rule would start a $fight$ complex event any time two punches are detected in a row with at least two different people appearing in the video in close proximity.

If we want to translate this example to the real world, however, there are some limitations. For starters, we need a system capable of identifying the simple events based on the CCTV feed. In ProbCEP, we rely on proxy models based on pre-trained neural networks. However, as these proxy models have been trained for another purpose, they are unable of providing such specific labels for the simple events. Furthermore, these models can be inaccurate, particularly if the input data is noisy or imprecise. For instance, a low-quality video could make it harder for the system to accurately detect when a fight is happening. In the following section, we define how we have emulated the definition from Equation 3.3 while using proxy models in ProbCEP.

3.2 Proxy Models Used

In order to detect violent situations in the UCF-Crime videos, our approach makes use of two pre-trained models as proxies. In this thesis, we consider two neural network models, as they seemed the most appropriate for the given task. However, it should be possible to use any approach that is able to accurately extract the required information from the input data. In the following sections, we describe these models as well as what their original functionality was designed to be. It is worth noting that, in order to use these models as proxies, these functionalities were slightly modified. These changes are described below in Section 3.3.

3.2.1 Activity Detector

The first system we consider is an activity detector, designed to identify different human-focused actions from short segments of video. For this purpose, we used the system presented in [Hara et al., 2018], which uses a 3D ResNet with 34 convolutional layers trained on the Kinetics-400 datasets, where it obtains an average accuracy of 0.71. When used for recognition, this system is designed to use a sliding window, where the video is split into non-overlapping 16-frame clips. This means that the video is split into many short clips, with a length of just over half a second—assuming the input video was recorded at 30 Frames Per Second (FPS). To recognize the activity happening in a given video, the system then gives each of these videos to the 3D ResNet and averages the class estimate outputs. The class that has the maximum score after averaging all of the clips is used as the prediction for the video.

Kinetics-400, which was used to train the 3D ResNet, is a large-scale labelled dataset of various human-focused actions. The dataset is composed of approximately 300,000 YouTube video URLs containing examples of 400 different classes, which mostly consist of human actions [Carreira and Zisserman, 2017]. The list of classes includes activities from many different areas, including sports (archery, boxing, tennis, ...), dancing styles (ballet, macarena, tap, ...), and cooking (cutting, frying, peeling, ...), among many others. However, it is worth noting that, while some of the classes may have a component that may be considered violent in certain scenarios (such as archery or boxing), none of the classes matches the type of violence we are trying to detect in UCF-Crime videos. The labelling for each video is decided automatically by inferring the activity from the title of the video, given by the uploader.

The ProbCEP approach assumes that the input simple events are instantaneous. While this is not possible if we want to identify how elements are moving as part of the simple events, we want the activity detection to look at very short periods of time, making it effectively instantaneous compared to the length of the complex events. This is the reason why we chose this 3D ResNet architecture, as it is capable of identifying a large range of different activities while using only a short fragment of video (about half a second).

3.2.2 Object Detector

We also use an object detector with the purpose of detecting relevant elements such as humans, animals and other objects in images. To this end, we consider Mask R-CNN [He et al., 2017], a framework for object instance segmentation. It is based on Faster R-CNN [Ren et al., 2015] and is designed to output a class label, a bounding box, and an object mask for each candidate object. Mask R-CNN consists of two stages. The first stage is a Region Proposal Network, which proposes potential object bounding boxes. In the second stage, Mask R-CNN predicts in parallel both the mask for the object, the class and the box offset.

For our approach, we use an implementation of Mask R-CNN which has been pre-trained using COCO, where it obtained an AP (Average Precision) of 39.8 on the task of object detection [Abdulla, 2017] (state-of-the-art at the time our experiments were performed). COCO [Lin et al., 2014] is a large-scale dataset that can be used for object detection, segmentation and captioning. The dataset contains 91 different classes with 330,000 images of complex everyday situations. COCO offers several features, including object segmentation and recognition in context. The classes of objects in the COCO dataset include people, multiple types of common objects, vehicles and animals, among others.

It is worth noting that other implementations of object detectors could be used in the place of Mask R-CNN. For instance, we have also tested using YOLOv3, which produced similar results. While more experiments would be required to ensure this, ProbCEP should, in theory, be capable of operating with any object detection system, as long as this system is capable of detecting the required classes. This is because ProbCEP is designed to give outputs based on the objects that have been detected and their positions, without relying on any implementation details of the approaches. As such, other object detectors could be considered if they are more accurate for the given task.

3.3 Our Approach: ProbCEP

ProbCEP is our approach to detecting complex events through the use of proxy models. In this section, we describe how we have used this approach with the models introduced above to

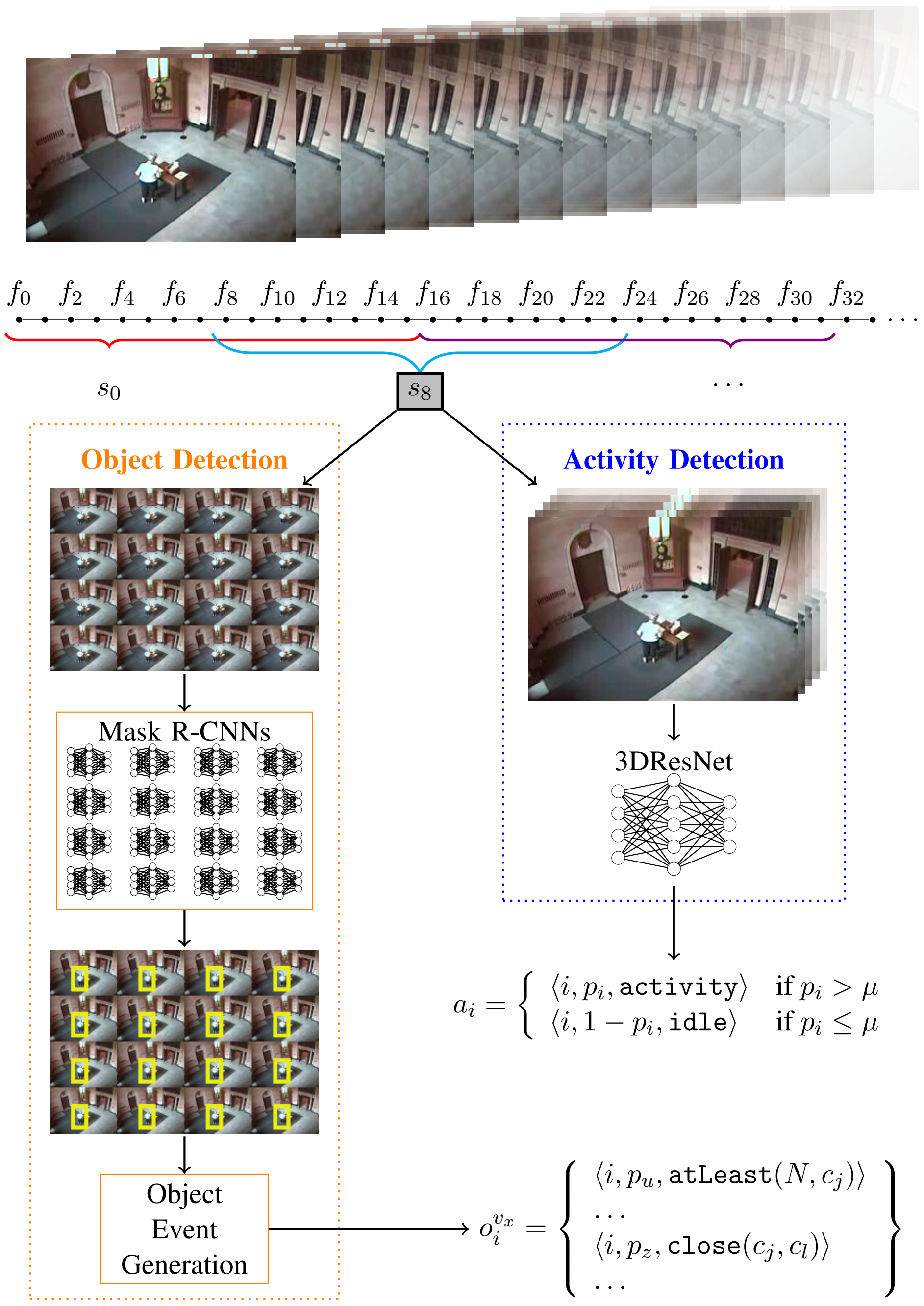

f_0 f_2 f_4 f_6 f_8 f_{10} f_{12} f_{14} f_{16} f_{18} f_{20} f_{22} f_{24} f_{26} f_{28} f_{30} f_{32} . . .
s_0
s_8
. . .
Object Detection
Activity Detection
Mask R-CNNs
3DResNet
Object Event Generation
$a_i = \begin{cases} \langle i, p_i, \texttt{activity} \rangle & \text{if } p_i > \mu \\ \langle i, 1 - p_i, \texttt{idle} \rangle & \text{if } p_i \leq \mu \end{cases}$
$o_i^{v_x} = \left\{ \begin{array}{l} \langle i, p_u, \texttt{atLeast}(N, c_j) \rangle \\ \ldots \\ \langle i, p_z, \texttt{close}(c_j, c_l) \rangle \\ \ldots \end{array} \right\}$

Figure 3.2: Data processing pipeline for detecting violence. Using frames from Abuse001 from UCF-Crime [Sultani et al., 2018] as an example.

detect violence from CCTV feeds. In order to do this, we defined the acts of violence we wanted to detect as complex events. Then, we used the proxy models to identify simple events from the input video. Finally, manually defined rules are used to detect when the complex events are occurring based on which simple events are happening. This allows the system to identify when violence occurs in the video. Figure 3.2 shows how the UCF-Crime videos are processed, using both the activity detector and object detector sub-systems. As the figure shows, each video is segmented in overlapping windows of 16 frames each, i.e. $\langle s_0, s_8, s_{16}, \ldots \rangle$. For any video where the number of frames was not a multiple of 8, the video was padded to fill the last segment. Each segment is then processed by both the activity detection and the object detection sub-systems, which produce the simple events used in the complex event definitions. The following sections provide a more in-depth description of how the activity detection and object detection sub-systems were implemented to generate the simple events (see Sections 3.3.1 and 3.3.2 respecitvely), as well as how the rules to define complex events are implemented based on these simple events (see Section 3.3.3). These following sections describe an approach similar to the one used in Example 1. However, due to limitations on the systems, the types of simple events are slightly different, which also results in a change to the rules.

3.3.1 Simple Events from Activity Detection

Algorithm 1 Activity Detector Simple Events

```
Input: $timestamps$, a list of the relevant timestamps. $\mu$, a threshold that determines above
    which value an *activity* simple event is generated (0.085 in our experiments).
Output: $activitySimpleEvents$, the list of simple events occurring based on the activity
    detector system.
1: $activitySimpleEvents \leftarrow$ *new List* ▷ Lists allow new elements to be appended at the end
2: for $t \in timestamps$ do
3:     $\langle actClass, conf \rangle \leftarrow$ *getActivityDetectionPredictionFor*$(t)$
4:     ▷ Get the most likely class for section $t$ (stored in $actClass$) and its confidence ($conf$)
5:     if $conf \geq \mu$ then
6:         $activitySimpleEvents$.*append*$(\langle conf,$ "*activity*"$, t\rangle)$
7:     else
8:         $activitySimpleEvents$.*append*$(\langle 1 - conf,$ "*idle*"$, t\rangle)$
9: return $activitySimpleEvents$
```

In order to generate the activity detection simple events, the 3D ResNet is used for each segment s_i, and simple events are generated based on its output. This is shown in Algorithm 1. Line 1 defines the list of activity detection simple events, where we will add all the generated simple events. Line 2 starts a loop over all the relevant timestamps. This will iterate over all the segments in a video, using the activity detection to process them. Line 3 obtains the most likely class ($actClass$) and its confidence ($conf$) from the 3D ResNet activity detection using the current segment as an input. While $actClass$ is returned, this is not relevant for the rest of the code. This is because the classes included in the Kinetics-400 dataset do not include terms that would be directly useful to us, such as the $punch$ event defined in Example 1. Instead, the output from the activity detection is used as a proxy in the following manner: if $conf$ is above a threshold $\mu = 0.085$ (empirically determined) we generate a simple event labelled *activity*, indicating that some activity is happening in the video. The probability of the *activity* simple event happening will be $conf$ (see lines 5 and 6). Otherwise, if all classes have a predicted probability below μ, a simple event labelled *idle* is generated instead, with a probability $1 - conf$ (lines 7 and 8). Finally the list of simple events generated is returned (line 9). Listing 3.1 shows a sequence of possible simple events that may be generated from the activity detection.

```
0.1933::happensAt(activity, 0).
0.9617::happensAt(idle, 8).
0.9748::happensAt(idle, 16).
0.1132::happensAt(activity, 24).
...
```

Listing 3.1: Example of possible simple events generated by the activity detection.

3.3.2 Simple Events from Object Detection

For each segment s_i, each of the 16 frames composing the segment is also individually passed to the object detector sub-system. The object detector is then used to identify which objects appear in each of those 16 frames. This serves as input to the Object Event Generation, as shown in Figure 3.2. This gives two types of outputs, further described in the following sections: (i) the probability of having at least N objects of each class in segment s_i; and (ii) the probability of having two objects close to each other within segment s_i. Listing 3.2 shows an example of a possible output given by the system. See the following sections for more details on how these values are calculated. Note that, for timestamp 0, there are three instances of the simple event `atLeast`. This is because each of them has a different confidence value. For instance, in this case, we are extremely confident that there is at least one person (98.72%) and very confident that there are at least two (89.10%). We also have an event saying that there might be three people in the segment, but it has very low confidence (9.50%). The low confidence indicates that this third person (real or miss-classified) is only detected in very few of the frames in s_0.

```
0.9872::happensAt(atLeast(1, person), 0).
0.8910::happensAt(atLeast(2, person), 0).
0.0950::happensAt(atLeast(3, person), 0).
0.9734::happensAt(close(person, person), 0).
...
```

Listing 3.2: Example of possible simple events generated by the object detection.

The main difference in the events detected by this object detection system and the ones shown in Example 1 is the fact that this approach only takes into account the classes in general, whereas Example 1 identified each object individually. This is because, in order to identify the same person or object across multiple frames and segments, a tracking algorithm would need to be added. Due to the complexity of keeping track of each object as it moves throughout the video, this has been left as future engineering. Instead, we consider only the objects appearing in each individual frame, making no assumptions to their relations to the objects detected in other frames.

In order to calculate the outputs for both types of object detection simple events, the system needs to detect all the objects in each of the segments, and group them appropriately. This is done following Algorithm 2, which returns an aggregation of dictionaries.

In this paragraph, we define our use of dictionaries in our algorithms. A dictionary $dict$ is a data structure that allows us to store *key: value* pairs. The keys in a dictionary must be unique (within one dictionary) and can be retrieved in the form of a list using the $dict.keys()$ function. Values can be assigned or retrieved from a dictionary by using the key as an index $dict[key]$. Dictionaries can be aggregated by using another dictionary as a value, with another set of keys. This could be used, for example, to store the average temperatures for each day in a year as follows $avgTemp[month][day]$. This aggregated dictionary would have the months of the year as a high level keys, and the days of each month as a lower level key. As such, $avgTemp[\text{“}Jan\text{”}]$ would give us a dictionary with the day numbers in January (1 to 31) as keys, and the average temperatures as values. It is also worth noting that the keys for the different sub-dictionaries do not need to be the same. For instance, in the average temperatures examples some months will have 31 days, whereas others will have less.

Algorithm 2 Split Objects By Timestamp

Input: $frames$, a list of all frames in the given video. $segSize$, the size of the segments in number of frames (16 for our experiments). $segPeriod$, the period between segment starts (8 in our experiments).

Output: res, an aggregation of dictionaries. From high to low level, the keys are the segments in the given video, the frame within that segment and the object classes appearing in that frame. The dictionary values is the list of detected objects for that class and that frame.

```
 1: function GETOBJECTSBYSEGMENT($frames, segSize, segPeriod$)
 2:     $res \leftarrow$ new Dictionary
 3:     ▷ Dictionaries have key-value pairs. The key can be used to consult or assign the value
 4:     for all $j \in frames$ do
 5:         $firstSegNum \leftarrow \lfloor (j - segSize + segPeriod)/segPeriod) \rfloor$
 6:         $firstSegNum \leftarrow max(0, firstSegNum)$        ▷ Ensure that $firstSegNum \geq 0$
 7:         $lastSegNum \leftarrow \lfloor j/segPeriod \rfloor + 1$
 8:         $objectsAtJ \leftarrow getObjectsAtTimestamp(j)$
 9:         for all $segNum \in [firstSegNum, \ldots, lastSegNum]$ do
10:             $segId \leftarrow segNum * segPeriod$
11:                                          ▷ The segment ID is the frame the segment starts on
12:             for all $object \in objectsAtJ$ do
13:                 if $segId \notin res.keys()$ then
14:                     $res[segId] \leftarrow$ new Dictionary
15:                 if $j \notin res[segId].keys()$ then
16:                     $res[segId][j] \leftarrow$ new Dictionary
17:                 if $object.class \notin res[segId][j].keys()$ then
18:                     $res[segId][j][object.class] \leftarrow$ new List
19:                 $res[segId][j][object.class].append(object)$
20:     return $res$
```

Algorithm 2 returns an aggregation of dictionaries that has the following keys (from high to low level): the segment s_i, the frames f_j within that segment and the object class. The value will be the list of objects in that segment and frame of that object class type. In order to generate this dictionary, the algorithm iterates over each frame in the video (line 4). For each frame, we find the segments it belongs to (lines 5 to 7). Due to the segment overlap, most frames will belong to more than one segment. In our experiments, all frames except for the first and last 8 will belong to two segments. However, this could change if segments of different sizes or a different segment period were used. We also detect all the objects in frame f_j (line 8). Then, for each segment s_i that the frame belongs to (line 9), we find the segment identifier (line 10) and add all the detected objects to their corresponding lists in the dictionary, based on the keys (lines 12 to 19). Lines 13 to 18 ensure that the required keys exist in the dictionary, and initialize them with an empty list if they do not. Line 19 adds each object to the correct list and finally line 20 returns the generated dictionary. In the following sections, we explain how this dictionary is used to calculate the probabilities for the AtLeast and Proximity simple events.

Instances of an Object Type in a Segment

In this section, we describe how the probabilities for the `atLeast` simple events are calculated. First, we split the detected objects into their corresponding segments according to the code in Algorithm 2.

Then, for each type of object appearing in segment s_i, we generate a set of `atLeast(m, c)` simple events. These simple events indicate the probability that there are at least `m` instances of the object class `c` in the given segment. These simple events are generated for each class appearing in at least one of the frames in the given segment. For each class `c`, the probabilities for the different values of `m` are calculated, with $1 \leq \texttt{m} \leq M$, where M is the maximum number of `c` objects in an individual frame from segment s_i. The function in Algorithm 3 is used to find M for each class `c`.

Algorithm 3 Get Max Occurrences by Object Class

Input: $objByFrame$ a dictionary with the objects detected in a segment. From high to low level, the keys for this dictionary are the frames within the segment and the object classes in each of those frames. The values are the lists of objects of that class detected in that frame.

Output: $maxOccByC$ a dictionary with the maximum number of occurrences for each class in $objByFrame$

```
1: function GetMaxOccByC($objByFrame$)
2:     $maxOccByC \leftarrow$ new Dictionary
3:     for all $j \in objByFrame.keys()$ do
4:         for all $\texttt{c} \in objByFrame[j].keys()$ do
5:             if $\texttt{c} \notin maxOccByC.keys()$ then
6:                 $maxOccByC[\texttt{c}] \leftarrow 0$
7:             $classObjInFrame \leftarrow length(objByFrame[j][\texttt{c}])$
8:             $maxOccByC[\texttt{c}] \leftarrow max(maxOccByC[\texttt{c}], classObjInFrame)$
9:     return $maxOccByC$
```

In order to generate these simple events, we first need to find $P(n \geq \mathtt{m})_j^{\mathtt{c}}$ for each frame f_j in segment s_i. That is, the probability that the number objects of class $\mathtt{c}$ in frame f_j is greater or equal to $\mathtt{m}$. We know that $P(n \geq \mathtt{m})_j^{\mathtt{c}} = 1 - P(n < \mathtt{m})_j^{\mathtt{c}}$. Since $P(n < 0)_j^{\mathtt{c}} = 0$, as there cannot be less than 0 objects for a given class, $P(n < \mathtt{m})_j^{\mathtt{c}} = \sum_{k=0}^{\mathtt{m}-1} P(n = k)_j^{\mathtt{c}}$. As such, $P(n \geq \mathtt{m})_j^{\mathtt{c}} = 1 - \sum_{k=0}^{\mathtt{m}-1} P(n = k)_j^{\mathtt{c}}$.

In order to calculate $P(n = k)_j^{\mathtt{c}}$, we need to consider the output from the neural network. Let $od_j^{\mathtt{c}}$ be the set of objects of class $\mathtt{c}$ detected in frame f_j, and $|od_j^{\mathtt{c}}|$ its cardinality—that is, the number of such objects. If $k > |od_j^{\mathtt{c}}|$, we know that $P(n = k)_j^{\mathtt{c}} = 0$, as there cannot be k objects in the frame (assuming the neural network has not missed any of them). If $k = |od_j^{\mathtt{c}}|$, we could say that $P(n = k)_j^{\mathtt{c}} = 1$, as the neural network has detected k objects. However, most object detection approaches give a confidence score for each object they detect, indicating how likely it is that the object is actually there. In some cases, the neural network may detect an object with low confidence. This indicates that such object may be there, but it may also be a false positive. As such, we should not assume that the number of objects detected by the neural network will always be exactly correct. If we treat these confidence scores as the probability that the object is actually there, the probability of having k objects in that f_j should actually be $P(n = k)_j^{\mathtt{c}} = \prod_{o \in od_j^{\mathtt{c}}}^{k} P(o)$. That is, the probability of there being k objects of that class in frame f_j is the product of all the confidence scores for the given objects. This also means that it is possible that there actually are exactly k objects of a given type, even if the neural network has detected more. This is because some of the additional instances may not actually be that object. As such, in order to calculate $P(n = k)_j^{\mathtt{c}}$ accurately, we need to find all the possible combinations where there are k objects and add their probabilities. We note the set of combinations $\binom{od_j^{\mathtt{c}}}{k}$. For each combination *comb*, we denote the set of objects we are considering present as $I = \{o \mid o \in od_j^{\mathtt{c}}, o \in comb\}$. We know that $|I| = k$. At the same time, $\bar{I} = \{o \mid o \in od_j^{\mathtt{c}}, o \notin comb\}$ is the set of objects we are considering as not present. Then, the probability $P(n = k)_{j,comb}^{\mathtt{c}} = (\prod_{o \in I} P(o)) * (\prod_{o \in \bar{I}} 1 - P(o))$. With this,

$$P(n = k)_j^{\mathtt{c}} = \sum_{comb \in \binom{od_j^{\mathtt{c}}}{k}} P(n = k)_{j,comb}^{\mathtt{c}}$$

which provides the general case for any $k \geq 0$, as both of the definitions for the special cases defined above can be derived from it. Specifically, if $k > |od_j^{\mathtt{c}}|$, then $|\binom{od_j^{\mathtt{c}}}{k}| = 0$, making $P(n = k)_j^{\mathtt{c}} = 0$. If $k = |od_j^{\mathtt{c}}|$, then $|\bar{I}| = 0$, making $\prod_{o \in \bar{I}} 1 - P(o) = 1$.

Algorithm 4 Get At Least Probabilities by Class, Number and Frame

Input: $objByFrame$ a dictionary with the objects detected in a segment. From high to low level, the keys for this dictionary are the frames within the segment and the object classes in each of those frames. The values are lists of the objects of that class detected in that frame. $maxOccByC$ a dictionary with the maximum number of occurrences for each class in $objByFrame$.

Output: $probAtLeastByC$ an aggregation of dictionaries containing the probability of having at least $\mathtt{m}$ objects of class $\mathtt{c}$ in frame j. From high to low level, the keys are the object class $\mathtt{c}$, the number of objects $\mathtt{m}$ and the frame in question j. The values will be $P(n \geq \mathtt{m})_j^{\mathtt{c}}$ for each $\mathtt{c}$, $\mathtt{m}$ and j in the given segment.

```
1: function GETATLEASTPROBS($objBySeg$, $maxOccByC$)
2:     $probAtLeastByC \leftarrow$ new Dictionary
3:     for all $j \in objByFrame.keys()$ do
4:         for all $\mathtt{c} \in maxOccByC.keys()$ do
5:             $probExactByK \leftarrow$ new Dictionary
6:                                  ▷ $probExactByK$ will contain $P(n = k)_j^{\mathtt{c}}$ for each $k$
7:             for all $k \in [0, \ldots, maxOccByC[\mathtt{c}] - 1]$ do
8:                 $probExactByK[k] \leftarrow 0$
9:                 for all $comb \in \binom{objByFrame[j][\mathtt{c}]}{k}$ do
10:                    $I = \{o \mid o \in objByFrame[j][\mathtt{c}], o \in comb\}$
11:                    $\bar{I} = \{o \mid o \in objByFrame[j][\mathtt{c}], o \notin comb\}$
12:                    $combProb \leftarrow (\prod_{o \in I} o.conf) * (\prod_{o \in \bar{I}} 1 - o.conf)$
13:                    $probExactByK[k] \leftarrow probExactByK[k] + combProb$
14:            if $\mathtt{c} \notin probAtLeastByC.keys()$ then
15:                $probAtLeastByC[\mathtt{c}] \leftarrow$ new Dictionary
16:            for all $\mathtt{m} \in [1, \ldots, maxOccByC[\mathtt{c}]]$ do
17:                if $\mathtt{m} \notin probAtLeastByC[\mathtt{c}].keys()$ then
18:                    $probAtLeastByC[\mathtt{c}][\mathtt{m}] \leftarrow$ new Dictionary
19:                $probAtLeastByC[\mathtt{c}][\mathtt{m}][j] \leftarrow 1 - \sum_{k=0}^{m-1} probExactByK[k]$
20:    returns $probAtLeastByC$
```

The function in Algorithm 4 is able to calculate $P(n \geq \mathtt{m})_j^{\mathtt{c}}$ for each $\mathtt{m}$, j and $\mathtt{c}$ in a segment, following the steps described above. Lines 5 to 13 calculate the values for $P(n = k)_j^{\mathtt{c}}$, and then lines 14 to 19 use those values to calculate $P(n \geq \mathtt{m})_j^{\mathtt{c}}$.

Once we have $P(n \geq \mathtt{m})_j^{\mathtt{c}}$ for each f_j, we need to calculate the aggregated probability for all the frames in segment s_i. This will be the probability for the simple event `atLeast(m, c)`. In order to calculate this, we use the expected value from a beta distribution. For that purpose, we calculate the accumulation of positive cases as $positive_i = 1 + \sum_{j=i}^{i+15} B * P(n \geq \mathtt{m})_j$, where B is a constant for the count. In our case, we used $B = 5$. On the other hand, the negative cases can be calculated as $negative_i = 1 + \sum_{j=i}^{i+15} B * (1 - P(n \geq \mathtt{m})_j)$. As such, the probability for the `atLeast(m, c)` simple event for segment s_i can be calculated as

$$P_i = \frac{positive_i}{positive_i + negative_i}$$

Algorithm 5 AtLeast Simple Events

Input: $frames$, a list of all frames in the given video. $segSize$, the size of the segments in number of frames (16 for our experiments). $segPeriod$, the period between segment starts (8 in our experiments). B, a constant used for the count (5 in our experiments), where $B \geq 0$. Higher values of B cause a more significant change to the probability for the simple events based on the same amount of evidence.

Output: $atLeastSimpleEvents$, the list of AtLeast simple events occurring based on the object detector system.

```
1: $atLeastSimpleEvents \leftarrow$ new List
2: $objBySeg \leftarrow \textit{GetObjectsBySegment}(frames, segSize, segPeriod)$
3: for all $seg \in objBySeg.\textit{keys}()$ do
4:     $maxOccByC \leftarrow \textit{GetMaxOccByC}(objBySeg[seg])$
5:     $probAtLeastByC \leftarrow \textit{GetAtLeastProbs}(objBySeg[seg], maxOccByC)$
6:                             ▷ $probAtLeastByC$ will contain $P(n \geq \mathtt{m})_j^{\mathtt{c}}$ for each c, m and $j$
7:     for all $\mathtt{c} \in probAtLeastByC.\textit{keys}()$ do
8:         for all $\mathtt{m} \in probAtLeastByC[\mathtt{c}].\textit{keys}()$ do
9:             $framesInSeg \leftarrow probAtLeastByC[\mathtt{c}][\mathtt{m}].\textit{keys}()$
10:            $positives \leftarrow 1 + \sum_{j \in framesInSeg} B * probAtLeastByC[\mathtt{c}][\mathtt{m}][j]$
11:            $negatives \leftarrow 1 + \sum_{j \in framesInSeg} B * (1 - probAtLeastByC[\mathtt{c}][\mathtt{m}][j])$
12:            $prob \leftarrow \frac{positives}{positives + negatives}$
13:            $atLeastSimpleEvents.\textit{append}(\langle prob, \text{“atLeast”}, \mathtt{c}, \mathtt{m}, seg \rangle)$
14: return $atLeastSimpleEvents$
```

Algorithm 5 shows how these calculations are performed for each possible m and c for each segment s_i in a given video. This code iterates over each list of objects detected for each segment, using the functions described in Algorithms 3 and 4 (lines 4 and 6 respectively) to obtain the values for each $P(n \geq \mathtt{m})_j^{\mathtt{c}}$. Then, lines 8 to 14 calculate the expected value from the beta distribution and generate the corresponding simple events.

Proximity

In this section, we describe how the simple events `close` are generated. An instance of these simple events is generated for each pair of object classes that can be found in the frames f_j of segment s_i. If two or more instances of the same class can be found in a single frame, a simple event for that class may also be generated for segment s_i. However, for each segment, only one instance of the simple event `close` will be generated for each pair of object classes.

Algorithm 6 Proximity Simple Events

```
Input: $frames$, a list of all frames in the given video. $segSize$, the size of the segments in
    number of frames (16 for our experiments). $segPeriod$, the period between segment starts
    (8 in our experiments). $D$, the length of the diagonal in the given video. $thresholdProb$,
    the probability threshold for proximity events (0.85 for our experiments). If a probability
    for a proximity event is lower than $thresholdProb$, the simple event is discarded.
Output: $proxSimpleEvents$, the list of proximity simple events occurring based on the ob-
    ject detector system.
 1: $proxSimpleEvents \leftarrow$ new List
 2: $objBySeg \leftarrow$ GetObjectsBySegment$(frames, segSize, segPeriod)$
 3: for all $seg \in objBySeg$.keys() do
 4:     $minDistance \leftarrow$ new Dictionary
 5:     for all $j \in objBySeg[seg]$.keys() do
 6:         $objClasses \leftarrow objBySeg[seg][j]$.keys()
 7:         $pairs \leftarrow \left(\!\binom{objClasses}{2}\!\right)$
 8:           ▷ Let $pairs$ be the set of possible 2-combinations with repetitions of $objClasses$
 9:         for all $p \in pairs$ do
10:             $\langle c1, c2\rangle \leftarrow p$                    ▷ Let $c1$ and $c2$ be the two elements of the pair $p$
11:             if $\langle c1, c2\rangle \notin minDistance$.keys() then
12:                 $minDistance[\langle c1, c2\rangle] \leftarrow \infty$
13:             for all $o1 \in objBySeg[seg][j][c1]$ do
14:                 for all $o2 \in objBySeg[seg][j][c2]$ do
15:                     if $o1 \neq o2$ then
16:                         $distance \leftarrow$ calculateDistance$(o1, o2)$
17:                         if $distance < minDistance[\langle c1, c2\rangle]$ then
18:                             $minDistance[\langle c1, c2\rangle] \leftarrow distance$
19:     for all $\langle c1, c2\rangle \in minDistance$.keys() do
20:         $prob \leftarrow 1 - minDistance[\langle c1, c2\rangle]/D$
21:         if $prob \geq thresholdProb$ then
22:             $proxSimpleEvents$.append$(\langle prob, "close", c1, c2\rangle)$
23: return $proxSimpleEvents$
```

Algorithm 6 shows how this simple events are generated. First, the list $proxSimpleEvents$ is defined, which will contain all the simple events generated (line 1). Then, the minimum distance between two objects for each pair of classes in each segment is calculated. In order to do this, the function form Algorithm 2 is used to get the grouped objects (line 2). The code then iterates over each segment (line 3) and defines a dictionary to keep track of the minimum distance between pairs of classes for that segment (line 4). The code then iterates over each frame in that segment (line 5), finding the object classes present in that frame (line 6). Line 7 calculates all possible pairs of object classes in the frame, with repetitions. This means that we are considering all possible pairs of different classes, as well as two instances of the same class. For each possible pair we define $c1$ and $c2$ to be the classes in the pair (lines 9 and 10). We ensure that this pair of classes is initialized in the dictionary (lines 11 and 12). Then, lines 13 to 18 iterate over all possible pairs of objects of those classes, finding the two closest ones in that frame. Line 15 ensures that we are not considering the same object twice, as the distance would then always be 0. Then, line 16 calculates the distance between the two objects. This is done based on the center of the bounding box, which is provided by the object detection neural network. If the distance is lower than our current minimum, the minimum is updated. After all the iterations, $minDistance$ will have the minimum distance between any two objects in any of the frames for the current segment, for each pair of classes that appear in the same frame. The distances between different frames are not calculated, as we do not know whether the camera is moving, which could change the results.

After this, we can generate the simple events based on the values in $minDistance$. For each pair of classes in the $minDistance$ keys, we calculate the probability of them being close in line 20. This uses the minimum distance divided by the diagonal size, which will give us a value between 0 and 1. This is because the furthest apart two objects can be is on either corner, making the maximum distance the size of the diagonal. We define that the probability of two classes being close is $1 - minDistance[\langle c1, c2\rangle]/D$. This will give a value of 0 if the objects are on opposite corners, and bigger values if the objects are closer. Finally, if this probability is bigger than a given threshold (line 21), the simple event is generated (line 22). This threshold is used as, with our current definition, two objects can be fairly far apart and still have high probabilities, which could affect the predictions in an unexpred manner.

3.3.3 Complex Event Definitions

In order to combine the outputs of the two sub-systems described above, we need to define the complex event rules. The approach is designed to allow users to use any set of rules that can be defined in ProbLog. For the results shown in this chapter, we use a re-implemented version of the framework shown in [Skarlatidis et al., 2015], which allows for the definition of complex events using an Event Calculus dialect. This has been done to provide compatibility with ProbLog2 [Fierens et al., 2015], as the original implementation was intended for ProbLog1 [De Raedt et al., 2007], which is not compatible with current versions of ProbLog. More specifically, we have implemented Equation 3.1 from the problem definition. This implementation can be seen in Listing 3.3.

```
holdsAt(F = V, T) :-
  initiatedAt(F = V, Tprev),
  Tprev < T,
  not broken(F = V, Tprev, T).

broken(F = V1, Ti, Tf) :-
  initiatedAt(F = V2, Tm),
  V1 \== V2
  Ti < Tm,
  Tm < Tf.
```

Listing 3.3: Prob-EC [Skarlatidis et al., 2015] implementation in ProbLog.

As such, we define that a complex event will be initiated each time two instances of the simple event *activity* occur in a row, whereas the complex event will terminate when two instances of *idle* happen in a row. Listings 3.4 and 3.5 show our implementation of such a definition. Such definitions of complex events would usually be specified by experts. However, in this case, we have empirically determined them based on our observations of the outputs for short sections of the videos.

Listing 3.4 defines the boundaries of complex event *videoOnly* on the basis of the activity detection system only. More specifically, lines 1 to 4 control the initialisation of the events and lines 6 to 9 control the termination. The initialisation checks that an *activity* event happens at the given frame (line 2) and then 8 frames later (lines 3 and 4), cf. Figure 3.2. The termination works in a similar manner but looks at the *idle* simple events instead of the *activity* ones.

This allows us to detect complex events based on the simple events produced by the activity

```
initiatedAt(videoOnly = true, T) :-
    happensAt(activity, T),
    Tnext is T + 8,
    happensAt(activity, Tnext).

initiatedAt(videoOnly = false, T) :-
    happensAt(idle, T),
    Tprev is T - 8,
    happensAt(idle, Tprev).
```

Listing 3.4: Anomaly detection on UCF-Crime with Prob-EC and data from the activity detection system only.

detection system. However, we can introduce further human knowledge into the system by defining further conditions. For example, in Listing 3.5 we show how the simple events from the object detector system can be used to define the complex event *videoObjDet*. This complex event also aims to identify violence, but it adds further conditions with the aim of reducing the number of false positives. The first rule (lines 1 to 4) defines that the complex event will be initiated if three conditions are fulfilled: (i) the complex event *videoOnly* is initiated at that timestamp—that is, two instances of *activity* happen in a row—(line 2), (ii) at least two people appear in the video (line 3) and (iii) those two people are close (line 4). This represents the human knowledge that two people need to be close to fight. Then, lines 6 and 7 define that the complex event *videoObjDet* will terminate at the same time *videoOnly* terminates.

```
1  initiatedAt(videoObjDet = true, T) :-
2      initiatedAt(videoOnly = true, T),
3      happensAt(atLeast(2, person), T),
4      happensAt(close(person, person), T).
5
6  initiatedAt(videoObjDet = false, T) :-
7      initiatedAt(videoOnly = false, T).
```

Listing 3.5: Anomaly detection on UCF-Crime with Prob-EC, fusing data from the activity detection and object detection sub-systems.

The system will output a value between 0 and 1 indicating the confidence it has that a complex event is occurring. According to the definition of the complex events, this confidence depends on the confidence for the *activity* and *idle* simple events, combined with the object detection simple events in the case of *videoObjDet*. Due to the large number of classes considered in the activity detection neural network, as well as the fact that none of the activities the neural network has been trained to detect actually occur in the videos we are considering, the confidence for the *activity* simple events tends to be quite low. As such, the confidence on the complex events also tends to be fairly low. For this reason, we will consider a complex event to be occurring if the confidence for that complex event is higher than 0.03, which has been empirically chosen. This value is high enough that a single initialization with low confidence of the *activity* simple events will not activate the complex event, possibly preventing false positives. At the same time, it is low enough that multiple initializations in a row, or even a single initialization from *activity* simple events with high enough confidence, can generate a complex event.

For the purposes of detecting complex events, we consider confidences above 0.03 to detect a complex event. This means that, if the system outputs a confidence lower than 0.03, we will consider that no complex event is occurring. Outputs of more than 0.03 will be considered as the system detecting a complex event. While the threshold of 0.03 may seem very low, it is worth keeping in mind that the input events from the activity detection can have very low confidences, meaning that getting higher confidences is not common.

3.4 UCF-Crime

To evaluate the performance of ProbCEP, we use UCF-Crime [Sultani et al., 2018], a dataset[1] comprised of 950 clips taken from CCTV footage, with each clip categorised as one of 13 classes of *abnormal* behaviour such as abuse, arrest, arson, assault, burglary, explosion, fighting, road accidents, robbery, shooting, shoplifting, stealing, and vandalism. UCF-Crime is also complemented with 950 *normal* videos where no such activities take place: the original intention was to create a larger dataset for anomaly detection.

The dataset also offers temporal annotations for a small subset of 140 anomaly videos, with different amounts for each of the classes. These are intended to be used as the testing dataset. For all other videos, only the overall label is provided, which simply indicates that the labelled activity happens within the video. However, the exact segment within which the abnormal behaviour happens in the video can vastly change from one video to the other. In fact, even within the test dataset, there are instances where the abnormal behaviour happens in two segments, although in most cases it happens in one continuous segment. The total length of this abnormal

[1]The dataset, including all the videos and some further details, is available for download at `https://webpages.charlotte.edu/cchen62/dataset.html` (on 27th of April 2022).

behaviour also varies a lot, in both total length and percentage of the video. In the test dataset, this varies from just 30 frames (1 second) to almost 7000 frames (close to 4 minutes), and from less than 1% of the video to 85% of it.

As explained in the introduction to this chapter (see Chapter 3), ProbCEP is an approach designed to detect complex events for which there is no training data or not enough time to train a system. As such, it is not designed to compete with systems that are trained specifically to detect the illegal activities from UCF-Crime and, therefore, we will not be comparing its performance with state-of-the-art approaches.

Instead, in our experiments, we show the results of our system for six of the videos, in the form of graphs. These graphs show the predictions for both the *Video Only*, which only makes use of the output from the activity detection, and the *Video + Object Detection*, which combines the outputs of the activity detection and the object detection. The graphs also show the ground truth of when the anomaly occurs in grey. The 0.03 threshold indicating when complex events are detected has also been added to the graphs as a horizontal dashed line.

Due to how we have defined our rules, it seems unlikely that our system would be able to detect actions that are purposely made in a discrete manner, such as shoplifting or stealing (which, in the dataset, seems to mostly consist of picking locks to steal vehicles). Similarly, the definition for the Video + Object Detection complex event would only take into account cases where people are close to each other. As such, cases where people are far apart (such as shooting) or cases that do not necessarily require multiple people in frame (such as arson or explosions) would not trigger this type of complex event. Of course, we could define new rules that correspond with those cases, but specifying rules for each of the cases would take a significant amount of time.

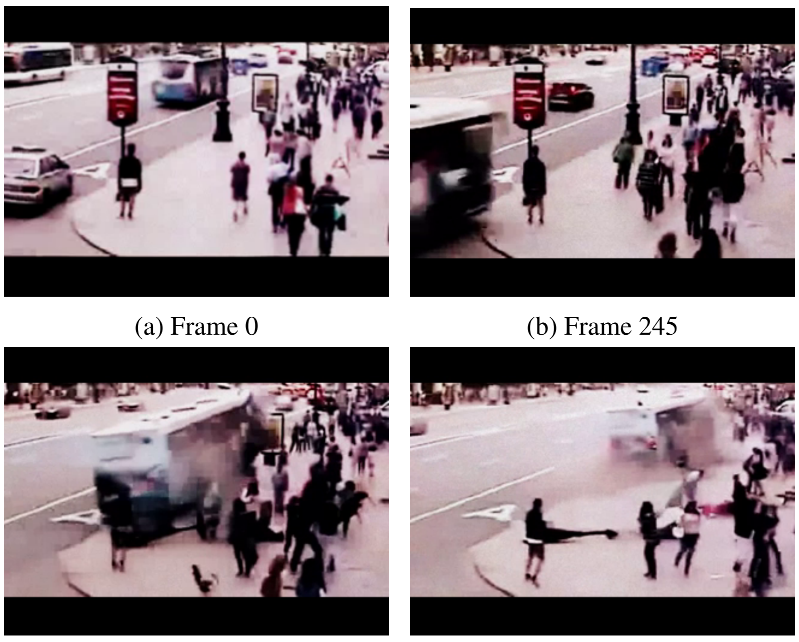
As such, we focus on exploring how the system performs in cases from the Fighting and Abuse classes. In general, videos classified as fighting involve two or more people, who tend to be fighting for longer periods of time. On the other hand, cases of abuse in the dataset tend to only take a few seconds, as they usually only consist of one punch or kick. This allows us to evaluate how the length of the abnormality affects the system's ability to detect it. We also look at cases of Road Accidents, in order to evaluate how the system performs in cases where the presence of people is not necessarily relevant to the abnormality.

3.5 Comparing Performance with [Sultani et al., 2018]

In this section, we compare the performance of the ProbCEP approach with the approach proposed in [Sultani et al., 2018] in one of the videos. Given the differences in the approaches, we would not expect ProbCEP to perform as well as [Sultani et al., 2018]. This is mainly due to the fact that ProbCEP does not perform any training, only relying on the proxy models that have been trained on data completely unrelated to what we are trying to detect and the manually defined rules. On the other hand, [Sultani et al., 2018] uses a multiple instance learning approach where the training videos from the dataset are divided in segments and put into bags, which contain normal and anomalous videos. That is, the training videos are put into anomalous and normal bags, where anomalous bags contain at least one segment that shows the anomaly, and normal bags only contain non-anomalous situations. Authors of [Sultani et al., 2018] then extract the features of each segment using a C3D and train a fully connected neural network to identify when an anomalous situation is shown in the video. Given that the approach from [Sultani et al., 2018] trains on the same type of data that it is trying to detect, we would expect it to perform better than ProbCEP, which only relies on proxy models.

Figure 3.3 shows some relevant frames from one of the videos from UCF-Crime. Particularly, it shows an example of a road accident, with the video titled RoadAccidents002. This video shows a street with a few cars and a bus moving. There is also a sidewalk with about 15 people walking (Figure 3.3a). About 7 seconds into the video, a bus enters the scene (Figure 3.3b) and almost immediately runs into a sign, a lamp post (Figure 3.3c) and a street advertising board (Figure 3.3d) before stopping.

The proposed approaches should be able to detect the anomalous part of the video. That is, they should be able to detect the section of the video where the bus runs over the sidewalk. Figure 3.4 shows the outputs for both approaches. For ProbCEP, it shows the results for both outputs: (i) Video Only and (ii) Video + Object Detection. These come from the different sets of rules defined above, namely Listings 3.4 and 3.5, respectively. As a reminder, the probability of the Video Only output increases when two *activity* simple events occur in a row, while it decreases when two *idle* simple events occur in a row. The probability for Video + Object Detection also decreases when two *idle* simple events occur in a row. However, in order to increase the probability for Video + Object Detection, we need to detect two people or more during a period


(a) Frame 0 (b) Frame 245

(c) Frame 280 (d) Frame 330

Figure 3.3: Some frames of RoadAccidents002 **from** **UCF-Crime** **[Sultani et al., 2018].**

of video, and those people need to be close together, on top of having two *activity* simple events occurring in a row. The generation of *idle* and *activity* simple events is based on the output of the pre-trained activity detector defined above in Section 3.2.1, while the detection of people is based on the pre-trained object detector described in Section 3.2.2.

Meanwhile, [Sultani et al., 2018] outputs a single value. For all three outputs we should expect a higher probability to indicate when the anomaly is occurring, with higher values indicating that the system is more confident in its prediction. On the other hand, systems should output lower values when no anomaly is occurring in the video. An ideal system would output 0.0 while no anomaly is occurring, and 1.0 when an anomaly is occurring, which would perfectly predict the period of time during which an anomaly is happening. However, due to how Prob-CEP has been implemented, such high probabilities are unusual, which is why we consider anything above 0.03 as the system predicting that an anomaly is occurring. This is indicated with a dashed line in the figures showing the results. On the other hand, for [Sultani et al., 2018] we should consider anything above 0.5 as the system detecting an anomaly, and anything below as normal (that is, no anomaly).

Figure 3.4 also has a grey box, indicating the period during which the anomaly is occurring

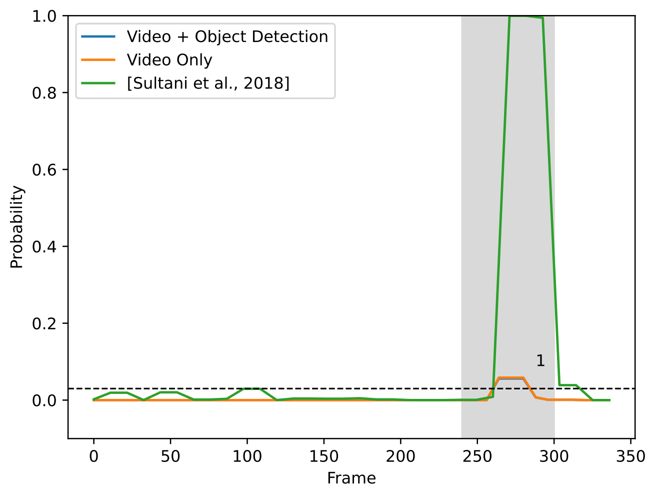


Figure 3.4: Results for RoadAccidents002**. 1 marks the period when the accident occurs.**

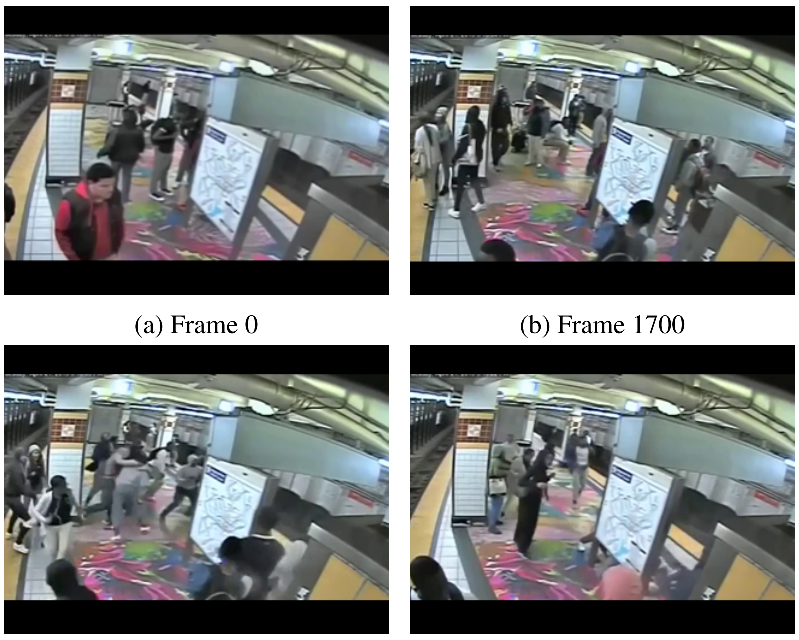
(labelled 1 in the figure), according to manual labelling. As indicated above, the predictions of a perfect system would exactly match this box. As expected, we can see that the prediction by [Sultani et al., 2018] has a much higher value than the outputs from ProbCEP (both for Video Only and Video + Object Detection). However, if we consider the lower threshold for ProbCEP, we can also see that both approaches detect the anomaly quite well. [Sultani et al., 2018] has some slight increases earlier in the video, but nowhere close to the 0.5 threshold for that approach. Interestingly, both approaches start detecting the anomaly at the same time, which is slightly later than the ground truth. This is likely because it is difficult to tell that the bus is going towards the sidewalk in the first few frames it appears until it starts crashing into things. The ground truth starts when the bus comes in, whereas the systems start predicting an anomaly once it starts crashing into things. With this in mind, we can consider ProbCEP to be fairly accurate for this case, taking into consideration the added limitations we have put into the approach compared to the [Sultani et al., 2018] approach.

However, we can also observe that the outputs for both Video Only and Video + Object Detection are effectively the same for the whole video, giving lines that are practically indistinguishable on the graph. This is despite the anomaly shown in the video being focused on the bus, and not in people fighting or similar. This is because the system is able to detect that there are multiple people walking on the sidewalk, thus fulfilling the condition of more than 2 persons being close to each other. Since the object detection system does not know where in the frame the anomaly is happening, this means that it is not able to identify if those people have anything to do with the potential anomaly. As such, it is worth keeping in mind that, although using Video + Object Detection can certainly remove false positives, as shown in the following section, this complex event may also be activated by people that have nothing to do with the actual abnormality.

3.6 Performance on Selected Videos

In the following sections, we describe each of the selected videos from UCF-Crime and provide an explanation of the predictions given by the ProbCEP system for them.

3.6.1 Fighting003



(a) Frame 0 (b) Frame 1700

(c) Frame 1870 (d) Frame 2550

Figure 3.5: Relevant frames of Fighting003 from UCF-Crime [Sultani et al., 2018].

Figure 3.5 shows some of the frames from Fighting003, a video showing the fight between multiple individuals happening in an underground station. The video starts with about 10 people waiting in the station (Figure 3.5a). During the first minute of the video, about a dozen other people enter the frame, with some of them also going out of view on the other end. One minute into the video, there are about 15 people in the frame (Figure 3.5b). Then, approximately half of them start fighting (Figure 3.5c), as the rest of passengers run away. The fight lasts until the end of the video.

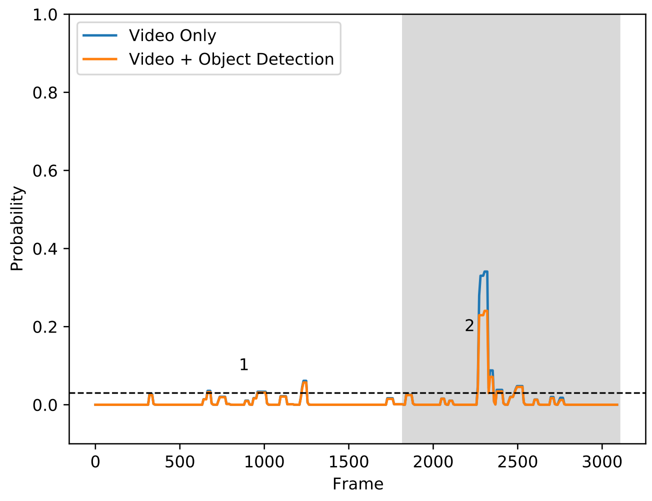


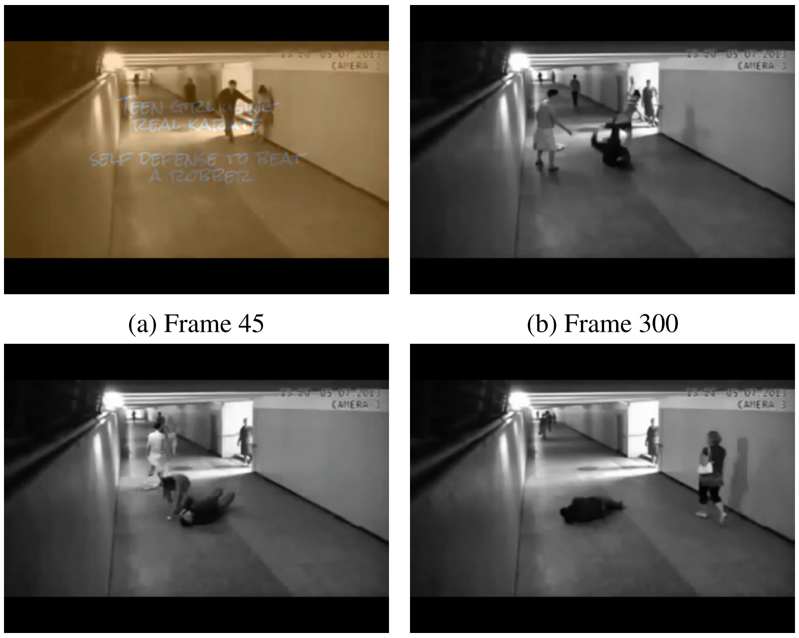
Figure 3.6: Results for Fighting003. 1 marks the period during which people are walking around in the station. 2 marks the period while they are fighting.

The results for this video are shown in Figure 3.6. This graph shows how the movement of people around the station generates some false positives (1 in Figure 3.6, which corresponds with the people moving over the first minute). However, the spike generated during the actual fight (2 in Figure 3.6, which corresponds to a particularly active moment in the fight) is much more substantial. Despite this, comparing against the ground truth (marked as a grey box in the graphs), a decent part of the fight is not being detected by the system.

3.6.2 Fighting018

Figure 3.7 shows some of the frames from Fighting018. This video is clearly edited, as it begins with a title section stating “Teen girl using real karate. Self defence to beat a robber.” This title then transitions to the view of an underpass, with about 5 people. As this transition is happening, we can see a robber stealing something from a girl (Figure 3.7a). The girl then starts fighting with the robber, punching and kicking him (Figure 3.7b), which knocks down the robber. Then, the girl recovers the stolen item (Figure 3.7c), and runs away. For the following 20 seconds of the video, the robber is laying on the floor as a few people walk past (Figure

3.7d). Finally, the video contains some more animated titles and a drawing of two people practising martial arts.



(a) Frame 45 (b) Frame 300

(c) Frame 380 (d) Frame 750

Figure 3.7: Some relevant frames of Fighting018 from UCF-Crime [Sultani et al., 2018].

Figure 3.8 shows the results for Fighting018. ProbCEP is able to detect the end of the fight (1 in Figure 3.8). However, it does not detect anything for roughly the first half of the fight. This is likely due to the fact that the body of the robber is obstructing the view, making it difficult to identify what is happening, even as a human. The system also incorrectly detects something just after the ground truth. Looking at the video, we can see that this corresponds with the time during which the girl is running away (2 in Figure 3.8). While this is technically not part of the fighting, it is still an abnormal behaviour, making it reasonable for the system to detect it, especially given that it has not been trained to detect any specific types of illegal activities.

In the latter part of the video, the system has slight increases as people walk past the robber on the floor (3, 4 and 5 in Figure 3.8), although none of them go past the threshold for detecting a complex event. Finally, the animations at the end of the video also generate a slight increase in the confidence for the complex event (6 in Figure 3.8), although the lack of people in the frame does prevent the Video + Object Detection complex event from triggering.

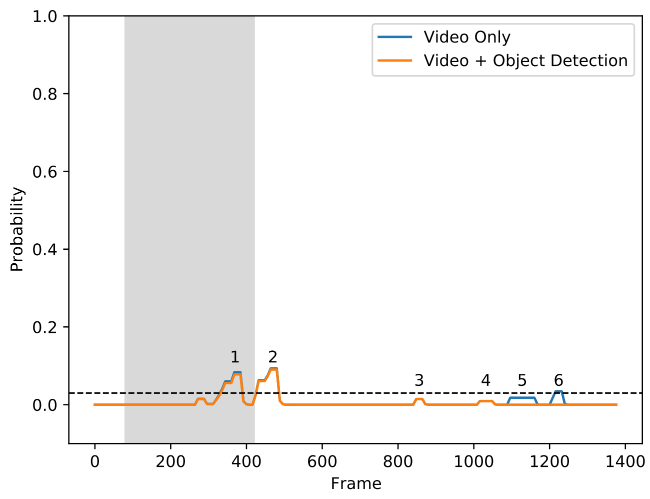


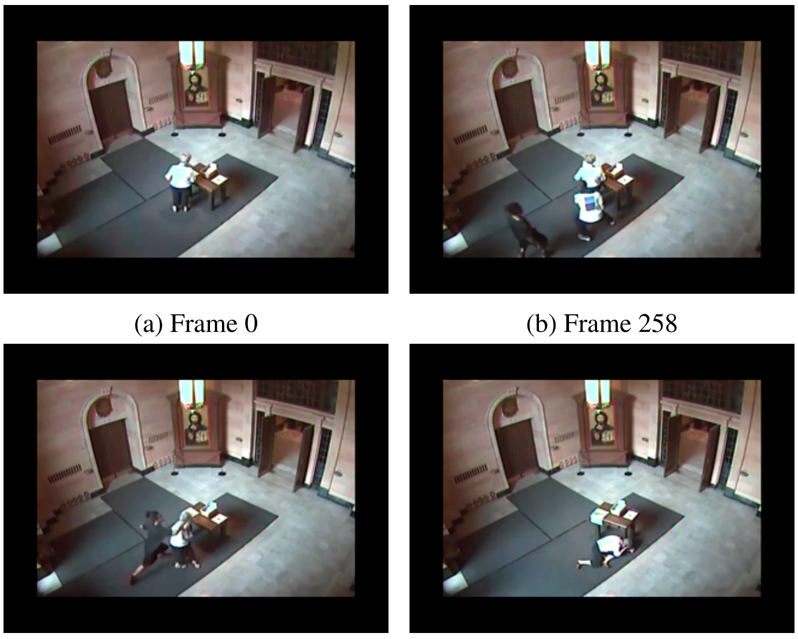
Figure 3.8: Results for Fighting018. 1 marks the spike generated by the end of the fight. 2 marks the spike generated by the girl running away. 3, 4 and 5 mark spikes generated while people are walking by. 6 marks the spike generated by the ending animation.

3.6.3 Abuse001

Figure 3.9 depicts four frames from Abuse001, one of the videos labelled as abuse. This video is a portion of surveillance video recording the north vestibule of St Cecilia Cathedral in Omaha, Nebraska, USA, where on 18th August 2015 CE, at about 11:06 am local time, a man struck a 76-year-old woman after another man stole her purse.[2] The video starts showing the woman alone in the church (Figure 3.9a), until the two men enter about 8 seconds into the video, and walk towards her (Figure 3.9b). Then, in quick succession, the first man steals the woman's purse, and the second man strikes the woman (Figure 3.9c). After that, the woman falls to the floor as the two men leave the church (Figure 3.9d). After a bit, the woman starts getting up. Approximately 25 seconds into the video, as the woman is getting up, the video loops back and is repeated 3 more times. However, as the content is the same in the following repetitions we will assume that the video ends just before the second iteration begins, making the video 25 seconds long.

In Figure 3.10 we see the results for this video. As shown in the graph, the system predicts

[2]Edited video is available on the Youtube channel of KETV NewsWatch 7—a local TV station—at `https://www.youtube.com/watch?v=uswCrbMymvE` (on 8th March 2019); the news story was reported also by the Omaha World Herald—a local newspaper—at `https://www.omaha.com/news/crime/video--year-old-woman-attacked-at-st-cecilia-cathedral/article_c60f4dcc-45de-11e5-bd6c-53122e3ec616.html` (on 8th March 2019).



(a) Frame 0 (b) Frame 258

(c) Frame 300 (d) Frame 380

Figure 3.9: Relevant frames of Abuse001 from UCF-Crime [Sultani et al., 2018].

when the abuse is happening very well, with both definitions (marked with 1 in Figure 3.10). The Video Only complex event is also triggered later in the video, which corresponds with the time when the woman is getting up (2 in Figure 3.10). However, the Video + Object Detection is able to filter this case out, as there is only one person in frame. As such, we can see that combining the outputs from both proxy models can be useful to filter out false positives.

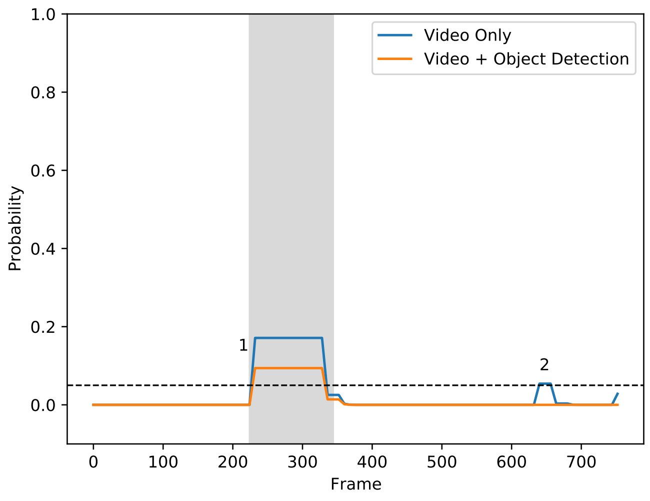
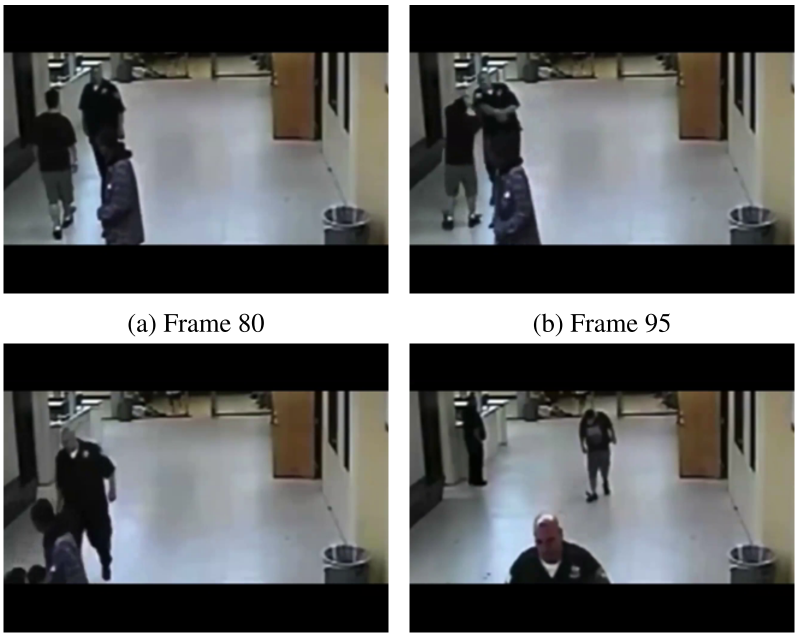


Figure 3.10: Results for Abuse001. 1 marks the spike matching the complex event of abuse. 2 marks the period when the woman is getting up.

3.6.4 Abuse007

Next, we evaluate how the system performs on Abuse007. Some of the frames for this video are shown in Figure 3.11. This video starts with two teenagers and what appears to be a guard walking in a hallway (Figure 3.11a). About 3 seconds into the video, the guard punches one of the teenagers (Figure 3.11b), who falls to the floor (Figure 3.11c). After this, the teenager gets up and both he and the guard walk up and down the hallway (Figure 3.11d), after which the video ends.



(a) Frame 80 (b) Frame 95

(c) Frame 125 (d) Frame 901

Figure 3.11: Some relevant frames of Abuse007 from UCF-Crime [Sultani et al., 2018].

Figure 3.12 shows the results for Abuse007. As shown in the graph, in this case, the abnormality is not detected. This is likely due to the fact that the actual act of abuse is very short, which does not give the system enough time to detect it. Despite this, we can see that there is a slight increase in confidence when the abnormality happens (1 in Figure 3.12). However, small spikes are also generated as the people in the video are walking around the hallway, which could cause false positives (2, 3, 4 and 5 in Figure 3.12). As such, it seems unlikely that complex events that happen in such a short span of time could be detected using this approach.

It is worth noting, however, that the spikes generated towards the end of the video (4 and 5 in the graph) do not generate a Video + Object Detection complex event, despite the fact that there are two people in frame for the whole video. This is thanks to the second part of the rule defined for this complex event, which specifies that those people must be close to generate a complex event.

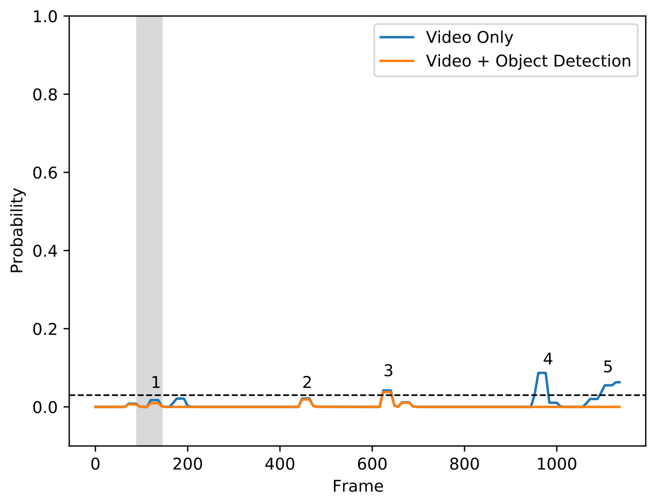


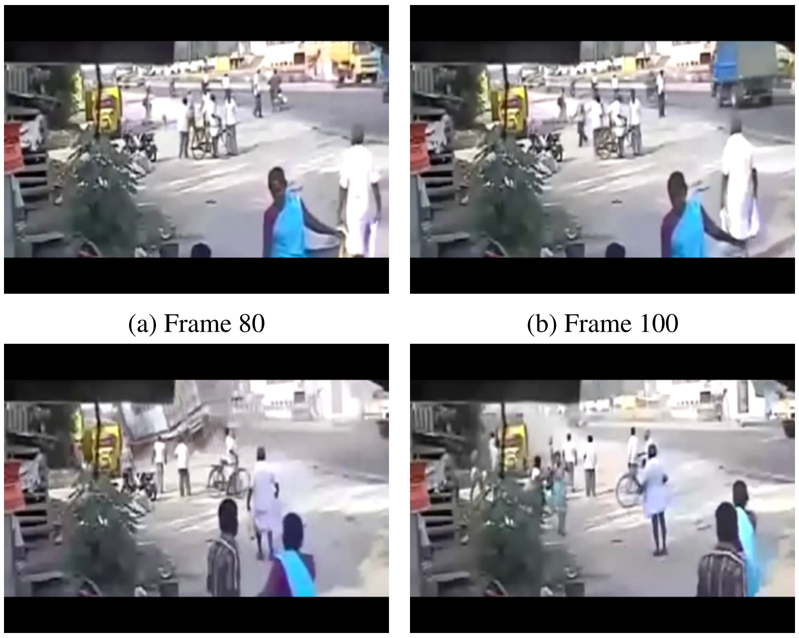
Figure 3.12: Results for Abuse007. 1 marks the act of abuse. 2, 3, 4 and 5 mark different instances where spikes are generated due to people walking in the hallway.

3.6.5 RoadAccidents110

Figure 3.13 shows some of the relevant frames from RoadAccidents110. The video starts showing a wide sidewalk with about 10 people on it, as well as a road in the background with people on motorbikes and some other vehicles (Figure 3.13a). About 4 seconds into the video, a blue truck enters the scene (Figure 3.13b), followed a second later by a white truck. This white truck immediately drives out of the street and onto the sidewalk, running over a pedestrian. The truck keeps going and starts to tilt to its left (Figure 3.13c) until it falls to its side (Figure 3.13d). The video ends a few seconds later as pedestrians start walking toward the truck, which is resting on its left side.

Figure 3.14 shows the results for this video. As the results show, there is no complex event generated during the time where the accident happens (3 in Figure 3.14). There are some slight increases in confidence while people are walking on the sidewalk (1 and 4 in Figure 3.14), but they do not go above the threshold for detecting a complex event.

Interestingly, despite missing the actual accident, the system does detect a complex event at the


(a) Frame 80 (b) Frame 100

(c) Frame 180 (d) Frame 300

Figure 3.13: Some frames of RoadAccidents110 **from UCF-Crime [Sultani et al., 2018].**

time where the blue truck enters the frame (2 in Figure 3.14). This may be due to the fact that the first truck is dark blue, which contrasts significantly with the clear tones of the area around it. As the truck appears suddenly and moving quickly, this could have triggered the complex event. On the other hand, the white truck that causes the accident may have been missed by the system because it blends in a lot more with the environment, making it harder to detect.

It is also worth mentioning that slightly changing the rules to, for instance, two *activity* simple events within any window of 16 frames (instead of the current 8) would detect this complex event, without adding too much noise to the other cases. This could be defined with the rules shown in Listing 3.6. While finding the best combination of rules to correctly detect the cases we are interested in would be difficult to do by hand, a possible area for further research could be automatically learning these rules using an inductive logic programming approach such as [Law et al., 2020], or other similar approaches.

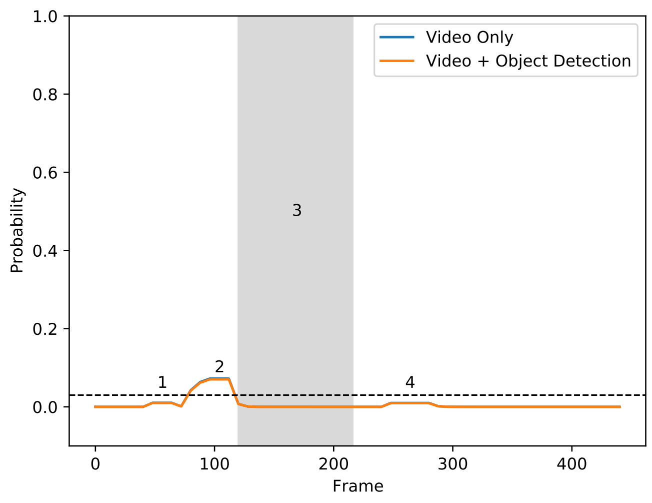


Figure 3.14: Results for RoadAccidents110. 1 and 4 mark different instances where people walking generate an increase in confidence. 2 marks the time when the blue truck enters the frame. 3 marks where the complex event should be detected.

```
initiatedAt(videoOnly = true, T) :-
    happensAt(activity, T),
    Tnext is T + 8,
    happensAt(activity, Tnext).

initiatedAt(videoOnly = true, T) :-
    happensAt(activity, T),
    Tnext is T + 16,
    happensAt(activity, Tnext).
```

Listing 3.6: Example of modified rules that increase the probability of `videoOnly` if two *activity* simple events happen within a window of 16 frames.

3.7 Improving Inference Speed to Allow Detection from Live Feeds

After investigating the previous results, the ProbCEP approach seems to have potential for detecting complex events using proxy models. However, we also found that using this approach on a live feed would result in some complications. We want ProbCEP to be applicable to live feeds, from which we can detect complex events in real-time. This would allow us to automatically detect if something happens in those feeds, and, for example, prompt a human to manually check what is happening. However, in order to be able to do this, the system needs to be fast enough to keep up with the input feed in real-time, as otherwise it will keep falling behind and end up becoming irrelevant due to the increased delay. There are, however, some difficulties due to the necessity of detecting complex events from a live feed.

The main difficulty comes from the fact that there is no endpoint at which we can execute the system and analyse a whole video. This means that, if we want live information, we need to perform the queries as new simple events come in, taking into account the information up until this point. This is because, unlike a video with a given end, there is no point at which we have all the information.

The most natural solution would be to perform a query on the new state of the complex events each time we get a new simple event. However, this approach has a problem when used for long periods of time. This is because the probability of a complex event holding at any point in time is based on three factors: (i) the probability of that complex event holding at the previous iteration, which will be (ii) increased for the current iteration based on the probability of that complex event being initiated and (iii) decreased based on the probability of that complex event being terminated. The probabilities for initiating and terminating will generally depend on the values for the recent simple events. However, the probability of that complex event holding on the previous iteration will recursively depend on the same three factors until the first timestamp being considered. Calculating the entire recursive call would not only be a huge waste of computation time, but would also require unfeasibly large amounts of memory usage. To avoid this, we made a slight modification to the system that keeps the probability of each complex event holding in the previous timestamp, which is provided to the logic layer each time a new value needs to be calculated.

Another issue, however, is the fact that such a large number of queries causes the system to be quite slow, to the point that it was struggling to keep up with 30 FPS videos (the framerate at which the example videos were recorded). This means that the system could keep falling behind from the real-time inputs, eventually making the outputs irrelevant due to how long ago they actually occurred. To improve the efficiency of the system, there are two areas in which we could focus; the proxy models and the logic inference. For the proxy models, we could consider using faster neural networks. For instance, we tried using the YOLOv3 object detector, which was faster than the Mask R-CNN system. However, faster models will not always be available, and would always be case-specific.

As a result, we decided to focus on improving the efficiency of the logic layer instead, which would benefit any ProbCEP application. In order to improve this performance, we first investigated which of the steps required to perform the logic inference required the most amount of time. In the following section, we explain the different steps and evaluate their time cost.

3.7.1 Default Performance Evaluation

Methodology

In order to perform a logic inference in ProbLog, there are various steps that need to be taken. Firstly, a query needs to be defined. This query takes the form of a clause, for which the model will output a probability of being true. The inference can then be started by combining a logic program with the query, which produces an *Arithmetic Circuit (AC)*. The process of obtaining the corresponding AC from a given logic program and query is described in more detail in Section 2.2.2, but can be summarized as grounding the program based on our query, thus removing any part of the program that is irrelevant for the question we are asking, converting this grounded program to an equivalent Boolean formula and finally transforming this Boolean formula into an AC. Once we have generated this AC, we can evaluate it, which provides us with the probability that our query is true. This is done by performing the arithmetic operations that form the AC, calculating the probability based on the input values.

To determine which of the steps takes the longest amount of time, we decided to use a profiler on the code. This told us how much time was required to process each line of code, thus allowing us to determine where the bottleneck of the code was. In order to perform this profiling, we execute the code to process a video that is 944 frames long (about 31.5 seconds). Since we are performing inference every 8 frames, this means that a total of 118 logical inferences are performed.

Results

Table 3.1 shows how much time is spent on each of the lines required to perform the logical inference for the given video. The lines shown in the table are executed each inference iteration, with the times shown being the accumulated time spent processing that line for the whole execution.

Line #	Time (μs)	Time %	Line Contents
1	166	0%	`query = 'query(holdsAt({event} = true, {timestamp})).\n'`
2	580	0%	`query = query.format(event=event, timestamp=timestamp)`
3	716	0%	`mdl_string = '\n'.join([str_model, upd_knowledge, query])`
4	123 469	1%	`model = PrologString(mdl_string)`
5	21 155 108	99%	`ac = evaluatable.create_from(model, semiring=s)`
6	57 792	0%	`evaluation = ac.evaluate()`

Table 3.1: Code used to perform ProbLog inferences, with the time spent on every line over one execution of ProbCEP.

Lines 1 and 2 prepare the query that needs to be performed. In this case, it is generated based on which complex event and timestamp we want to evaluate. Then, the query is combined with the rest of the logic program (lines 3 and 4). This includes the information on the previous value for the complex event holding, as well as the values for the relevant simple events.

Then, line 5 obtains the AC from the given logic program and query by grounding the program, transforming it into a Boolean formula and finally into an AC, following the steps described in Section 2.2.2. Finally, the AC is evaluated, returning the probability of the given query being true (line 6).

As shown in Table 3.1, most steps are very quick, including the last step of performing the operations to calculate the result of the AC. This is because, once we have obtained the AC, we only need to perform addition and multiplication operations. However, the process required to obtain this AC is quite time-consuming. As shown in the table, this takes up 99% of the time required to perform the inference. As such, this may seem like the obvious part of the system to optimize. However, weighted model counting is a well-studied problem, and it would be difficult to get any significant improvements in the performance.

There is, however, another option. As we have seen, evaluating the AC once it has been generated is very quick, compared to the time required to generate it. As such, if we could re-use the ACs, instead of generating them every iteration, we could significantly improve the time performance of the system. In the following section, we explain how we have used this approach to improve the time-efficiency of our system.

3.7.2 Our approach: PreCompilation

Due to the nature of our system, most of the ACs will have a similar format. This is because, as explained above, the probability of a certain complex event holding depends on the probabilities of that complex event being initiated or terminated, as well as the probability of that complex event holding on the previous iteration.

We can calculate the probability of the complex event being initiated or terminated based on the rules defined by the user. While different rules may be true at different points in time, the complex event rules defined by the user will remain the same throughout the whole execution. This means that the ACs for this part of the program will be similar for each iteration. Meanwhile, as explained above, the probability of the complex event holding on the previous iteration will be provided to the system, in order to avoid repeating the entire recursive call each iteration.

As such, we can generate a single AC and use it for each iteration. This would improve the time-efficiency of the approach in a very significant manner, as it would allow us to almost entirely remove the time required for compiling. However, there are some difficulties with this approach. These are mostly due to the fact that, despite being similar, there are some important differences between the ACs for each query.

Firstly, the timestamp for the queries and, consequently, the timestamps we are evaluating for the initialization and termination rules will be different. For example, in the case of detecting violence from CCTV feeds described above, the query *holdsAt(videoOnly = true, 16)* will depend on the simple events that happen at timestamps 0 and 8. Meanwhile, the query *holdsAt(videoOnly = true, 32)* will depend on the simple events from timestamps 16 and 24. This will cause issues as the names of the nodes in the AC will not match correctly. However, we can solve this issue using identifiers. As long as the timestamps for the simple events required to perform the query $\{T_{se_1}, \ldots, T_{se_N}\}$ are defined in relation to the timestamp for the query T_q using a fixed difference, the timestamps can be converted automatically from one query to the other. That is, assuming that $T_{se_n} = T_q - \Delta_n$ for all n between 1 and N, we can find the value of each Δ_n given T_q and T_{se_n}. With this, if we then perform another query at $T_{q'}$, we can calculate that the values of $T_{se'_n} = T_{q'} - \Delta_n$, allowing us to know which simple events from q correspond with the simple events for q'. Following the example, both queries only care about the simple events that happened 16 and 8 timestamps ago. With this, we can transform both queries to a constant identifier, and then transform them back, allowing us to have one single AC. This approach would not work if the rules took into account the absolute value of the timestamp. For instance, it would not work if different rules were applied after timestamp 1000. However, the intention of using the system on a live feed will usually be to start monitoring a feed that may have been running for an arbitrary amount of time. As such, it seems unlikely that rules taking into account how long the system has been running would be used.

A second issue is that the simple events happening on the previous timestamps may be different. For starters, the probability of each simple event happening is almost certainly going to be different from one timestamp to another. Following the example above, the probability of the simple event *activity* happening at timestamps 0 and 8 will, in effectively every case, be different from the probability of it happening at timestamps 16 and 24. This issue, however, is fairly easy to solve. After transforming the timestamps to the constant identifiers described above, we can simply change the probabilities on the corresponding nodes of the AC for each simple event. Since these are the input values of the AC, performing the operations on the new values will provide us with the correct answer for the new query.

There is, however, a bigger problem with the fact that different types of simple events may happen at different timestamps. For instance, at timestamp 0 we may have an *activity* event, followed by an *idle* event at timestamp 8. However, at both timestamps 16 and 24, we may have an *activity* event. The issue comes from the fact that, in the first step of the compilation, ProbLog grounds the program, removing any part that is irrelevant to the given query, in order to simplify the compilation process as much as possible. This means that, in this step, the program will recognize any rules that are impossible for the given query and remove them before performing any further steps on the compilation. As a result, in our example, for the query *holdsAt(videoOnly = true, 16)*, ProbLog would recognize that the event cannot be initiated or terminated, as two instances of *activity* or two instances of *idle* are required to fulfil the corresponding rules, respectively. As such, ProbLog would identify that the probability of the complex event holding would only depend on the value from the previous iteration, making the AC very simple. Meanwhile, for the *holdsAt(videoOnly = true, 32)*, ProbLog would recognize that the complex event can be initialized, but not terminated. As such, this would generate different ACs depending on the combination of simple events that have happened recently. One possible solution to this would be to generate an AC to account for every possible combination of inputs. However, this would quickly become infeasible if the number of types of simple events increases. A more generic approach is to generate an AC that takes all the combinations into consideration at the same time. This can be archived by generating the circuit when all the simple events happen at all the possible timestamps. For our running example, this would mean generating the AC we wish to re-use in a situation where **both** simple events *activity* and *idle* happen both 16 and 8 timestamps before the query. This will generate an AC that will take into consideration both the initiation and termination rules. Finally, in order to use the AC in cases where a certain simple event does not happen, the corresponding probability simply needs to be set to 0 in the AC, which will provide us with the same result as if that part of the circuit did not exist.

By applying the solutions described above, it is possible to re-use a single AC for each type of query, thus removing the need to perform the compilation each iteration. Algorithm 7 shows a summarized version of the operations to perform in order to evaluate queries using the Pre-Compilation approach. In order to implement those solutions, however, we need to know the format of the queries that will be performed, as well as which inputs may be used to evaluate those queries. This can be deduced by looking at the ProbLog code and the corresponding rules. However, this needs to be done manually, as it is not possible to generate a system that can automatically detect what knowledge will be provided.

In order to facilitate the process of generating the ACs to be re-used, we have created a Python library called PreCompilation[3]. This library can be used in any ProbLog program that performs a similar query repeatedly. As such, it could be useful in other situations than just for improving the speed of ProbCEP. For more details on how to use this library, see Appendix A.

LiveEvents

LiveEvents is an implementation that simplifies the use of the PreCompilation approach when used to perform CEP using ProbCEP. LiveEvents is designed to allow the user to detect complex events from a live feed of simple events using the ProbCEP approach. Furthermore, it

[3]Code available at `https://github.com/MarcRoigVilamala/ProbLogPrecompilation`

Algorithm 7 Summary of the PreCompilation approach.

Input: $baseQuery$, the query required to generate the re-usable AC. It should generate a generic query that can be used with any other query to be performed. $updateableParameters$, a list of parameters that may need to be updated to perform other queries. $queriesToPerform$, a list of queries that need to be performed. This should contain both the format of each query as well as all the input values that need to be considered for that query.

Output: res, a list containing the results for each of the queries in $queriesToPerform$.

```
1: $preCompAC \leftarrow \textit{compileAC}(baseQuery)$
2: $weightsToUpdate \leftarrow \textit{findWeightsFor}(preCompAC, updatableParameters)$
3: $res \leftarrow \textit{new List}$
4: for all $query \in queriesToPerform$ do
5:     $preCompAC.\textit{updateWeightsFor}(weightsToUpdate, query)$
6:     $currentQueryRes \leftarrow \textit{getResult}(preCompAC, query)$
7:     $res.\textit{append}(currentQueryRes)$
8: return $res$
```

makes it easy to use PreCompilation, as it has already implemented the specifics for the CEP case. At the same time, it still allows the user to perform the queries without using the PreCompilation approach, thus allowing us to compare which effect PreCompilation has on the time performance of the system. As such, we will use LiveEvents to evaluate the effectiveness of PreCompilation when using ProbCEP to perform CEP.

3.7.3 Approaches to Compare

In this section, we describe the approaches that are compared in our experiments. For this purpose, we consider two approaches: (i) basic iterative approach and (ii) PreCompilation approach. It is worth noting that these approaches are designed to be able to provide results each iteration, making it possible to give a prediction each time new information comes in (assuming they are fast enough). This is in contrast to the case of evaluating at the end, which can only provide results once all the information has been obtained. While evaluating at the end would be faster than the basic iterative approach overall, this approach is not useful for evaluating from a live feed, as there is no point in time where we have all the information.

Basic iterative approach In the basic iterative approach, we compile an AC each iteration and use it to perform the inference to obtain the results, following the steps shown in Section 3.7.1. This results in a large amount of compilations, which significantly slow down the process.

PreCompilation approach The second approach is to PreCompile the ACs, and modify their values to perform the inference that provides us with the results, as described in Section 3.7.2. While these modifications do take a small amount of time, this is almost imperceptible when compared to compiling the AC, resulting in a significant increase on the speed of the system.

On the other hand, the fact that the re-usable AC needs to take all possible circumstances into account may make it more complex than the AC generated for a single iteration for the other approaches. As such, the time required to perform a single query may be higher. Despite this, the faster inference time once the AC has been compiled should make up for this as the number of queries grows larger.

In the following sections, we will present three experiments and their results using both approaches. The first two experiments focus on simple generic ProbLog programs, using independent and recursive queries respectively, with the aim to show that the PreCompilation approach can work on any ProbLog program that performs a similar query a large number of times. The third experiment explores how the PreCompilation approach improves the performance in LiveEvents.

For all experiments, a machine with an Intel Core i9-10900K with 10 cores and 62 GB of RAM was used. All experiments have also been performed three times, with the results shown being the average of the three executions.

3.7.4 Experiment 1: Independent Queries

In order to evaluate independent queries, we have used the example shown in Listing 3.7, where the query `twoHeads(T)` will only be true if both the inputs `heads1(T)` and `heads2(T)` are true. Different values of `T` can be used to represent different likelihoods for the inputs. In order to perform the experiment, we have created inputs for all values of `T` between 0 and 250, for both `heads1` and `heads2`. In each case, a random likelihood is assigned to both inputs.

```
twoHeads(T) :- heads1(T), heads2(T).
```

Listing 3.7: ProbLog code used to test independent queries.

Algorithm 8 Code used to calculate the time to perform different numbers of queries using both approaches.

Input: $maxQueries$, an integer indicating the maximum amount of queries to test.
Output: res, a dictionary indicating the required time for each number of queries for each approach.

```
1: $res \leftarrow$ new Dictionary
2: for $approach \in \{\text{“}Basic\text{”}, \text{“}PreCompilation\text{”}\}$ do
3:     $res[approach] \leftarrow$ new Dictionary
4:     for $nQueries \in [1, \ldots, maxQueries]$ do
5:         $startTime \leftarrow now()$
6:         $performNQueriesWith(approach, nQueries)$
7:         $res[approach][nQueries] \leftarrow now() - startTime$
8: return $res$
```

In order to compare the time performance for both approaches when performing different numbers of queries, we used Algorithm 8. As shown in the code, both approaches are used to perform N queries, for N between 1 and `MAX_QUERIES` (which was set to 250 for our experiments). Then, the amount of time required to perform each number of queries is recorded. It is worth noting that the `performNQueriesWith` function performs all the queries from scratch each time, without saving any results between different calls.

Results

Figure 3.15 shows how the PreCompilation approach is clearly superior in this case, as it takes the same amount of time in the worst case, and is significantly faster in other cases. As such, in this case there is effectively no drawback to using the PreCompilation approach, whereas there are significant benefits when the number of queries is large.

3.7.5 Experiment 2: Recursive Queries

In the second experiment, we perform recursive queries, where the result of each query depends on the result of the previous query. In order to evaluate recursive queries, we used the example shown in Listing 3.8, where the probability of the query `atTime(T)` depends on the inputs

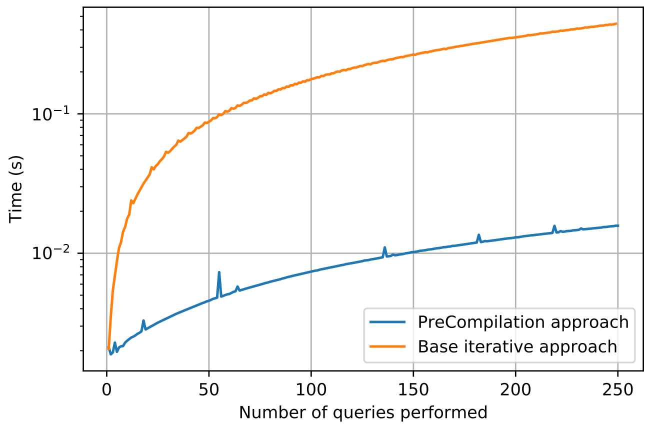


Figure 3.15: Graph showing amount of time required to perform different number of independent queries with each approach. Note the logarithmic scale on the vertical axis.

`increase(T)` and `decrease(T)`. This is a simplification of the rules used for ProbCEP. The probability of `atTime(T)` being true will depend on the value at the previous iteration (`atTime(Tprev)`). If `increase(T)` is true with a certain probability, the probability for `atTime(T)` will increase (line 1). Meanwhile, if `decrease(T)` is true, the probability will decrease from the previous iteration (line 2).

```
1 atTime(T) :- increase(T).
2 atTime(T) :- Tprev is T - 1, Tprev >= 0, atTime(Tprev), \+decrease(T).
```

Listing 3.8: ProbLog code used to test recursive queries.

In order to find the time results for this approach we use a similar code to the one used for experiment 1, as shown in Algorithm 8. The maximum number of queries is again set to 250.

Results

Figure 3.16 shows the performance of both approaches when using recursive queries. In this case, the PreCompilation approach is still clearly the fastest approach for large numbers of queries. However, focusing on very small numbers of queries (less than 5) there is a small period where the basic iterative approach is slightly faster. This is because, in order to work correctly in all cases, the AC generated in the PreCompilation approach needs to be slightly more complex. In the following section we explain why this is the case, along with visualizations for the corresponding ACs.

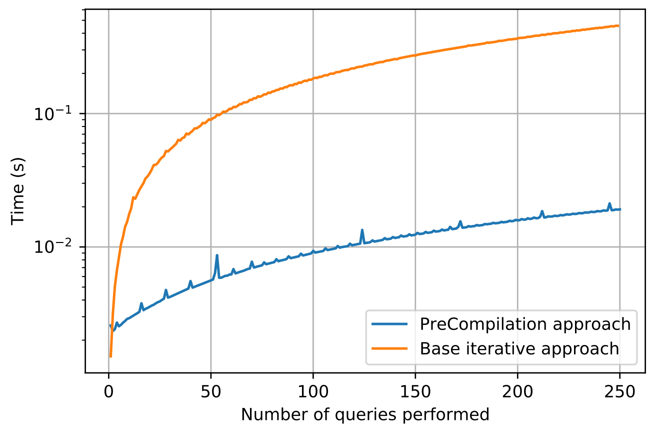


Figure 3.16: Graph showing amount of time required to perform different number of recursive queries with each approach. Note the logarithmic scale on the vertical axis.

```
0.747997::decrease(0).
0.147508::increase(1).
0.461236::decrease(2).
```

Listing 3.9: Input to generate example ACs.

Visualization of the Arithmetic Circuits

After applying the process described in Section 2.2.2 to a given logic program and query, we can obtain the corresponding AC. Any given AC has a set of inputs, which are combined using arithmetic operations to provide an output value. For our case, the inputs are the probabilities of our ground facts (i.e. the probabilities associated to `increase(T)` and `decrease(T)` each iteration), and the arithmetic operations are generated based on the logic rules we have defined to ensure that the output (i.e. the probability of our query being true) is logically consistent with the inputs, according to those rules. In the ACs generated by ProbLog, the arithmetic operations are represented by `AND`, `OR` and `NOT` nodes. For our purposes, these are substituted by the operations of *multiplication*, *addition* and *1 minus the given value*, respectively, which means that performing the arithmetic operations that constitute the AC will output the probability of the query being true[4].

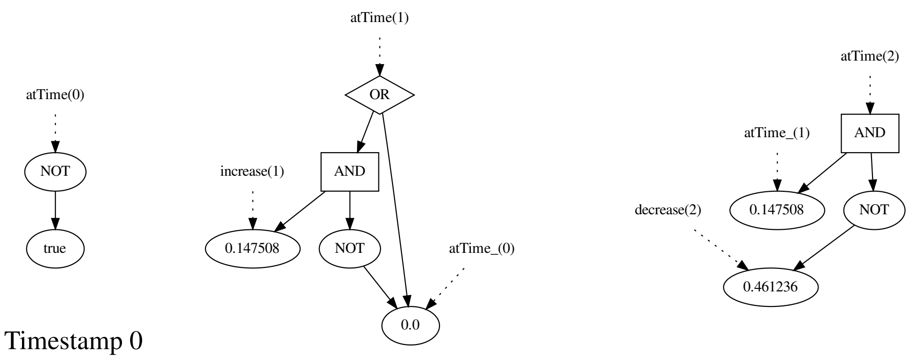
[4]Other substitutions are possible, allowing the same AC to be used to solve other problems, such as which is the most likely word in which the query is true, or how many worlds there are where the query is true. An interested reader is referred to [Kimmig et al., 2011]

ACs can be visualised in the form of a graph, where each node is either one of the inputs or one of the operations. Then, values can be propagated through this graph by performing the operations in the nodes, which will produce the output value after the results for all the operations have been calculated. Figure 3.17 shows some of the ACs generated by the basic iterative approach, using the input shown in Listing 3.9. More specifically, it shows the ACs generated to perform the queries for timestamps 0 to 2. Figure 3.17a shows the AC generated for timestamp 0, where the query performed is `atTime(0)`. Since the first line of our input clauses (Listing 3.9) is `decrease`, there is nothing making the query true. As such, the graph in Figure 3.17a simply shows that the value for `atTime(0)` is NOT true —that is, false. Note that the dashed arrows assign a label to a node. Depending on the label, this can either represent the result of the query (such as `atTime(0)` in Figure 3.17a) or one of the input values (such as `increase(1)` in Figure 3.17b). Meanwhile, solid arrows indicate that the value of a node is used in the operation shown on another node. For example, in Figure 3.17a the node `true` is used in the `NOT` operation.

The AC for timestamp 1 (see Figure 3.17b), shows how the input nodes `increase(1)` and `atTime(0)` at the bottom left and right, respectively. This is because the values for both of these can affect the value of our query `atTime(1)`. The operations shown in the graph have been generated by ProbLog to provide the correct output probability for the query, according to the rules. In this case, `atTime(1)` will be true if `atTime(0)` is true or if `increase(1)` is true. In the graph, however, the `increase(1)` part is considered inside an `AND` clause that also includes a `NOT(atTime(0))`. This is done by ProbLog to ensure that the two probabilities considered in the top level `OR` (labelled `atTime(1)`) are mutually exclusive, thus making it possible to calculate that value by simply adding both of them.

Finally, Figure 3.17c shows the AC generated for timestamp 2, where we have a `decrease` clause once again. In this case, however, the system needs to take into account the probability from the previous iteration, making the AC slightly more complex than in timestamp 0. More specifically, the graph shows that `atTime(2)` will be true if `atTime(1)` is true and `decrease(2)` is not true —that is, false.

While the ACs for the basic iterative approach become slightly more complex when they need to take into account the previous iteration, in all cases they are significantly simpler than the



(a) Timestamp 0

(c) Timestamp 2

(b) Timestamp 1

Figure 3.17: Arithmetic circuits generated for each iteration using the base iterative approach in the recursive experiment.

one generated for the PreCompilation approach, shown in Figure 3.18. This is because, in the PreCompilation case, the reusable AC needs to be able to represent all of the cases without modifying the shape of the graph. This means that the graph from Figure 3.18 can be equivalent to all three graphs shown in Figure 3.17, as well as any other graph that may be generated by the iterative approach. This can be done by substituting the values for the nodes in Figure 3.18 labelled `atTime(0)`, `increase(1)` and `decrease(1)` with the appropriate values for each case. To do this, each of the labels needs to take the value of their corresponding label in the original graphs (taking the difference in timestamps into account). For example, the graph for timestamp 1 (Figure 3.17b) has the labels `atTime(0)` and `increase(1)`. If we copy those values into the graph shown in Figure 3.18 (while leaving `decrease(1)` at 0.0), both graphs will provide the same output for `atTime(1)`. Similarly, we can copy the input values from timestamp 2 (Figure 3.17c) —that is, labels `atTime(1)` and `decrease(2)`— into the reusable graph, which will now give the result corresponding to timestamp 2. In this case, however, we need to take into account the difference in timestamps on the query, meaning that the values need to go into the labels `atTime(0)` and `decrease(1)` from Figure 3.18, respectively, with the output appearing on the node labelled `atTime(1)`. Thanks to the fact that the graph shown in Figure 3.18 can represent all the graphs that can be generated using the basic iterative approach (those shown in Figure 3.17 and more), using the PreCompilation approach we only need to generate that AC one time, and then we can reuse it to evaluate any combination of inputs. Since we know the values of the inputs, and all the possible clauses that

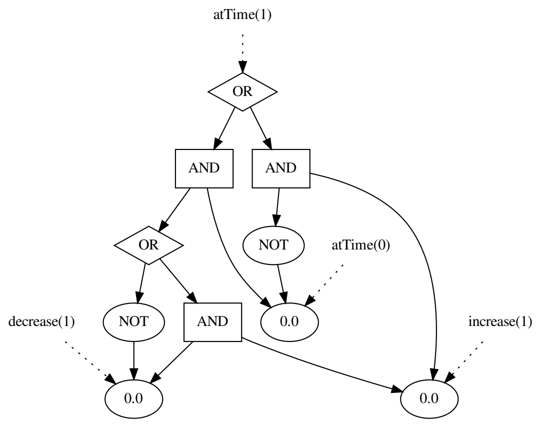


Figure 3.18: Arithmetic circuit used for the PreCompilation approach in the recursive experiment.

could be used as an input, we do not need to generate an AC every iteration, vastly increasing the inference speed when a large number of queries is performed.

Due to the higher complexity of the reusable AC, the PreCompilation approach is slightly slower when performing very small amounts of queries. However, for this example, the PreCompilation approach is more efficient when performing more than 3 queries, making it far better for large amounts of queries. The exact point at which using PreCompilation becomes more efficient will mainly depend on the difference in complexity between the ACs generated in the other approaches and the generic AC generated for the PreCompilation approach.

3.7.6 Experiment 3: LiveEvents

In order to evaluate the effectiveness of the PreCompilation approach on ProbCEP, we compare how using it affects the time performance of the system using LiveEvents. For this purpose, we take into account two values: (i) initialization time and (ii) frames per second.

Initialization time refers to the time required to start the system. This includes the time required to set up the input feed and the outputs. More importantly, however, we want to know the difference that using PreCompilation makes on the system. Of course, we would expect that using PreCompilation requires a larger initialization time, as the ACs need to be pre-compiled

	Base	**PreCompilation**
Initialization time (s)	**0.00011**	0.03288
Frames per second	211.48	**4729.85**

Table 3.2: Comparison of initialization time and processed frames per second depending on whether PreCompilation is used or not.

for the queries we want to perform.

Frames Per Second (FPS) refers to how many frames the system is able to process each second. In order to allow the system to keep up with a live feed, this needs to be at least the same frame rate as the input video (usually 30 FPS). However, larger values would still allow us to process more feeds in parallel.

In order to obtain those values, we have used a video file containing a total of 3566 frames, with the inference of the logic rules being performed every 8 frames. In order to remove as many variables as possible, we do not perform inference on the neural networks in this experiment. Instead, the approaches use pre-generated outputs, which allow us to better see the effect that the PreCompilation approach has on the speed of the logic layer.

Results

Table 3.2 shows how much time the system takes in the initialization stage, as well as the frames per second. As shown in the table, the PreCompilation approach does take a significantly longer time to perform the initialization, as expected. However, it more than makes up for it with the number of frames per second it can process. In fact, if the system is trying to process frames as quickly as possible, even if the video was 1 second long (30 frames), using the PreCompilation approach would already be an improvement in time-efficiency.

It is worth noting that, under the assumption that the neural networks take effectively no time at all to calculate the values, the base approach would still be able to keep up with a live feed from a CCTV camera, as the videos used in the example above use 30 FPS. However, using PreCompilation would still be useful if multiple CCTV feeds need to be processed at the same time. For this case, the base approach would barely be able to process 7 feeds at the same time, whereas the PreCompilation approach could process over 150.

3.8 Discussion and Conclusion

This chapter has presented ProbCEP, an approach that uses proxy models to detect complex events from non-symbolic data streams. As shown above, ProbCEP can be used to detect violence from CCTV feeds, achieving a reasonable amount of success even with simple definitions for the complex events. We have also shown that combining multiple proxy models can improve the performance of a system. For instance, defining rules on the detection of people allowed us to remove false positives in the predictions for some of the videos. We have also shown that, through the use of PreCompilation, it is possible to increase the speed of the ProbCEP approach to allow it to process live CCTV feeds. This is made possible by creating a generic AC, which can then be re-used, making it possible for the system to skip the compilation step of the process of performing a logical inference.

Despite this, there are some limitations to the ProbCEP approach. In particular, depending on the proxy model used, it can be difficult to obtain enough information from them to correctly identify the complex events, particularly if we want to differentiate between different types of complex event. Of course, this would not be as significant of an issue if the proxy models used provided more information about the specific situations we are trying to detect. However, in the context of proxy models trained using completely unrelated datasets, the amount of information that can be extracted using manually defined rules is limited.

Another potential limitation of the ProbCEP approach is the availability of proxy models. In some cases, proxy models may not be available at all, making it impossible to use the ProbCEP approach. In other cases, there might be models available but these may not be appropriate to use as proxy models. This could be, for example, because the situations being considered are slightly different, making the model perform significantly worse as a proxy model. While having a model that has an excellent accuracy would intuitively mean that it should perform well as a proxy model, as it should be able to differentiate the simple event classes very well, this may not always be the case. For instance, we may have an object detector that is trained on very high-resolution images, where it performs very well. However, this same model may perform very badly when detecting objects from a low-resolution CCTV feed. This will change on a case-by-case basis, depending on what proxy models are being used, and what input streams they will be used on. If some labelled simple events for the input streams being used are available, it may be possible to evaluate the accuracy of the proxy model in those cases. Otherwise, some analysis of the data will be required to figure out how effective the proxy models will be.

In the following section we explore how feasible it is for the ProbCEP approach to be able to differentiate the classes from UCF-Crime based on the output of the proxy models and, more specifically, the activity detection predictions. Meanwhile, the following chapters focus on approaches that do not require proxy models to be available, using training data to train the neural networks as part of the system instead.

3.8.1 Limitations

Being able to differentiate between the abnormality classes in UCF-Crime would be desirable, as they are likely to require different responses depending on which type of activity is occurring. A possible approach to differentiate them would be to rely on the objects detected in the frame. For instance, we may be able to differentiate between abnormalities where people must be in frame (such as stealing, fighting, ...) and those that do not (road accidents, explosion, ...). However, most activities would be difficult to differentiate based solely on the objects that appear in them. This is because most of the classes can happen in similar scenarios without requiring a specific object. Furthermore, even in the cases were an object is required, that object is not necessarily exclusive to that class. For instance, some type of gun needs to appear in a video to be classified as shooting. However, it would also be reasonable to see guns in cases of arrest or robbery, even if they are never shot. As such, while some differentiation may be possible through what objects appear in frame, it will not allow for a full classification. This is also made more difficult due to the fact that the definitions provided by the creators of UCF-Crime are not necessarily mutually exclusive. As humans, it is generally fairly easy to differentiate between cases of road accidents and arson, or even between arrests and fighting. The differences, however, are much more subtle when having to decide between burglary, robbery, stealing or shoplifting, or when trying to determine whether a video belongs to abuse or assault.

While experts would likely be able to classify these based on their legal definitions, defining rules that automatically classify them based on the outputs of our proxy models would be difficult, or even impossible. This is particularly true for the objects appearing in the videos, as there is always a risk that such objects appear in the background, without having any connection to the actual anomaly. For this reason, in this section we evaluate if it may be possible to differentiate between the different anomalies in UCF-Crime based on the amount of *activity* simple events generated by the videos that contain them. In order to evaluate this, we evaluate the Poisson distributions of the number of *activity* simple events occurring in the videos for each of the assigned labels l_i (which can be Abuse, Arrest, Arson, etc.). We have chosen to use a Poisson distribution as it is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time.

For that purpose, lets suppose we have the video $v_x^{l_i}$ where x is the identifier for the video. As defined above, l_i is the assigned label for the video. Then, we define $\phi_x^{l_i} = |\{\langle i, \xi\rangle \mid \exists a_i^{v_x} = \langle i, p_i, \xi\rangle, p_i > \mu\}|$, where μ is the threshold used to determine whether an *activity* or an *idle* simple event should be generated from the activity detection output. That is, $\phi_x^{l_i}$ is the number of *activity* simple events detected by the activity detection subsystem in the video $v_x^{l_i}$. Given a label l_i, let $\boldsymbol{\phi}^{l_i} = \langle \phi_1^{l_i}, \dots, \phi_{|\boldsymbol{v}^{l_i}|}^{l_i}\rangle$ be the vector listing the occurrences of simple events between the videos with weak label l_i.

We can then define Φ^{l_i} to be the number of occurrences of simple events in videos weakly labelled l_i, $\Phi^{l_i} \sim Poisson(\lambda_{l_i})$. As a prior, we assume a gamma distribution, as it is a conjugate prior for the Poisson distribution, meaning that the posterior distribution is also a gamma distribution computable in closed-form. More specifically, let us assume as prior $\Lambda_{l_i} \sim Gamma(\alpha_{l_i}, \beta_{l_i})$, with $\alpha_{l_i} = \beta_{l_i} = \epsilon$ arbitrary small constant (in the following $\epsilon = 10^{-100}$). Hence,

$$\Lambda_{l_i} \mid \Phi^{l_i} \sim Gamma\left(\alpha_{l_i} + \sum_{j=1}^{|\boldsymbol{v}^{l_i}|} \phi_j^{l_i},\ \beta_{l_i} + |\boldsymbol{v}^{l_i}|\right) \tag{3.4}$$

Figure 3.19 depicts the gamma distributions of the various λ_{l_i} for each of the 13 weak labels l_i in UCF-Crime, evidencing that there is a correlation between the number of simple events in videos and their weak labels. The gamma distribution for each of the classes has a peak at the expected value, indicating that the density at that point is the highest. As such, the gamma distributions show the likelihoods for a range of numbers of occurrences of the *activity* simple

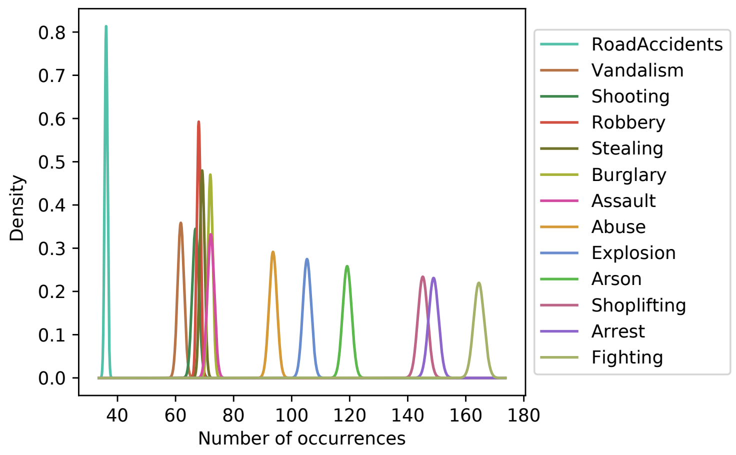


Figure 3.19: Gamma distributions of λ_{l_i} for each weak label l_i of UCF-Crime. Labels in legend sorted by expected values.

events for videos in each class. The rather constrained values for standard deviations support the assumption of homogeneity. That is, the number of simple events in videos with the same weak label l_i is homogeneous across all the videos.

However, we can also see that this correlation is not necessarily unique. From visual inspection of Figure 3.19 it is evident that some of the classes can be uniquely identified, namely *RoadAccidents*, *Abuse*, *Explosion*, *Arson* and *Fighting*. However, the rest of the classes show some overlap, which is quite significant in most cases. This includes the classes of *Vandalism*, *Shooting*, *Robbery*, *Stealing*, *Burglary*, *Assault*, *Shoplifting* and *Arrest*. This means that, while it may be possible to differentiate the first set of classes based on the occurrence of *activity* simple events, differentiating the remaining eight would be significantly harder.

Despite this, we also explore the distribution of the combination we are using as the start of a complex event; that is, two *activity* simple events in a row. For a video $v_x^{l_i}$, let

$$\widehat{\phi}_x^{l_i} = |\{\langle i, \xi \rangle \mid \exists a_i^{v_x} = \langle i, p_i, \xi \rangle, p_i > \mu \text{ and } \exists a_i^{v_x} = \langle i-8, p_{i-8}, \xi' \rangle, p_{i-8} > \mu\}|$$

i.e., the number of occurrences of detecting two *activity* simple events consequently via the activity detection subsystem in the video $v_x^{l_i}$. Given a label l_i, let $\widehat{\boldsymbol{\phi}}^{l_i} = \langle \widehat{\phi}_1^{l_i}, \ldots, \widehat{\phi}_1^{|\boldsymbol{v}^{l_i}|} \rangle$ be the vector listing the occurrences of simple events between the videos with weak label l_i. Then, let

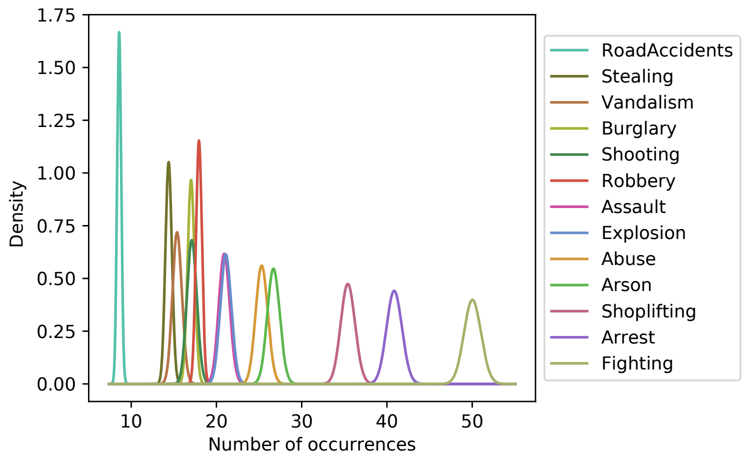


Figure 3.20: Gamma distributions of $\widehat{\lambda}_{l_i}$ for each weak label l_i of UCF-Crime. Labels in legend sorted by expected values. The colour for each class is consistent with Figure 3.19.

$\widehat{\Phi}^{l_i}$ be the number of occurrences of two consequent simple events in videos weakly labelled l_i: $\widehat{\Phi}^{l_i} \sim Poisson(\widehat{\lambda}_i)$. From (3.4), we then have:

$$\widehat{\Lambda}_{l_i} \mid \widehat{\Phi}^{l_i} \sim Gamma\left(\alpha_{l_i} + \sum_{j=1}^{|\boldsymbol{v}^{l_i}|} \widehat{\phi}_j^{l_i},\ \beta_{l_i} + |\boldsymbol{v}^{l_i}|\right) \tag{3.5}$$

Figure 3.20 depicts, for each of the 13 UCF-Crime weak labels l_i, the Gamma distributions of the various $\widehat{\lambda}_{l_i}$. In this case, some of the classes are still clearly differentiable from the others. This includes *RoadAccidents*, *Shoplifting*, *Arrest* and *Fighting*. The rest of the classes, however, have significant overlapping with each other. Furthermore, in the case of *Assault* and *Explosion*, the distributions are almost the same, having the same standard deviation and only a difference of 0.2 in the expected value. This is despite the fact that those two classes had significantly different expected values on the number of *activity* simple events λ_{l_i}.

Table 3.3 shows the expected values and standard deviation for the gamma distributions for both λ_{l_i} and $\widehat{\lambda}_{l_i}$. The expected values represent the weighted average for each class, meaning that most cases should have values around this expected value, slightly lower or slightly higher. Meanwhile, the standard deviation indicates the dispersion of those values, with high standard deviations indicating high dispersion and low standard deviations indicating that the values are more localized. Table 3.3 confirms what the graphs show, with the low standard deviations indicating that the classes are homogeneous. The expected values also confirm that some classes are differentiable, but some would be harder to differentiate. On top of that, Table 3.3 also shows that neither λ_{l_i} nor $\widehat{\lambda}_{l_i}$ correlate with the length of the videos in the dataset, lending further credibility to the fact that their values depend on the class displayed on the videos, and not on how long those videos are.

Class	λ_{l_i}		$\widehat{\lambda}_{l_i}$		Num Frames	
	E	**SD**	**E**	**SD**	**E**	**SD**
Abuse	93.6	1.4	25.3	0.7	483.2	579.3
Arrest	148.9	1.7	40.9	0.9	742.9	1127.0
Arson	119.1	1.5	26.7	0.7	679.2	2179.5
Assault	72.1	1.2	20.9	0.6	324.3	358.0
Burglary	72.0	0.8	17.0	0.4	588.5	843.4
Explosion	105.3	1.5	21.1	0.6	630.5	2463.1
Fighting	164.5	1.8	50.0	1.0	646.7	724.1
RoadAccidents	36.1	0.5	8.6	0.2	216.8	267.1
Robbery	68.0	0.7	17.9	0.3	351.6	313.3
Shooting	66.9	1.2	17.1	0.6	368.2	441.5
Shoplifting	145.2	1.7	35.4	0.8	810.4	1309.5
Stealing	69.2	0.8	14.4	0.4	583.7	551.2
Vandalism	61.8	1.1	15.4	0.6	367.4	332.8

Table 3.3: Expected value (E) and standard deviation (SD) of λ_{l_i}, $\widehat{\lambda}_{l_i}$, and number of frames in videos (mean and standard deviation of the sample) for each of the 13 l_i in UCF-Crime, in alphabetical order. The colours on each row are consistent with the legends for Figures 3.19 and 3.20.

On the other hand, it is worth noting that *Shoplifting* and *Arrest*, which had a significant overlap on λ_{l_i}, have no overlap on their $\widehat{\lambda}_{l_i}$ distributions. As such, it may be possible to differentiate between all the different classes using the appropriate combinations of simple events. These combinations, however, would be difficult to find manually. As such, it seems unlikely that rules can manually be defined to identify which type of abnormal activity is happening in the given CCTV feed.

3.8.2 Future Work

Given the type of problem we are trying to solve and our objectives, we believe that there are two main approaches that could be used to improve the accuracy and utility of the system.

Both of these approaches aim to maintain a similar architecture to ProbCEP, where a neural network is combined with symbolic rules, as this provides us reliability by letting us know what the complex event rules are, and agility by letting us change them. However, unlike ProbCEP, these approaches aim to learn how to detect the complex events in a more accurate manner. While this does mean that some amount of training data is required for either of such approaches, both approaches should be able to learn using sparse data, making them feasible even in situations where the amount of data is limited.

The first option would be to learn the rule definitions based on the output of the proxy models. As discussed above in Section 2.2.1, there are different approaches that are capable of learning complex event rules. Some approaches include [Margara et al., 2014, Bruns et al., 2019]. This could allow us to find definitions for the complex events that better represent when they are happening, as well as allowing us to automatically differentiate the types of abnormality happening in the video. This would be particularly useful in situations where humans are not able to define the rules for the complex events. For instance, it might be able to automatically find combinations of simple events that allow the system to differentiate between each of the UCF-Crime classes. In the area of CEP, however, the approaches that automatically learn rules generally do so directly from the input data. As discussed in Section 2.2.1, this type of approach is not intended for use on non-symbolic data, meaning that they could not be used in the type of situations we are considering. There is, however, some research into combining pre-trained neural networks and Inductive Logic Programming (ILP) [Cunnington et al., 2021a, Cunnington et al., 2021b]. While this research is not currently focusing in the area of CEP, it seems likely that it could be used to learn the rules for the complex events. According to [Cunnington et al., 2021a], one of the challenges of this type of approach is the fact that the neural networks may incorrectly predict the class for a given input, possibly with high confidence. This can make it difficult to correctly learn the rules, as ILP approaches try to generate a hypothesis—that is, a set of rules— that encompass all the training cases, or at least most of them. In order to correctly learn the rules, approaches try to reduce a *penalty* value. This value generally depends on the number of cases that are not encompassed by the current hypothesis, and the length of such hypothesis. This is done with the intent of tending towards simple rules, as they tend to be more understandable and less over-fitted for the training cases. Figure 3.21 shows a simplified architecture for both types of approaches.

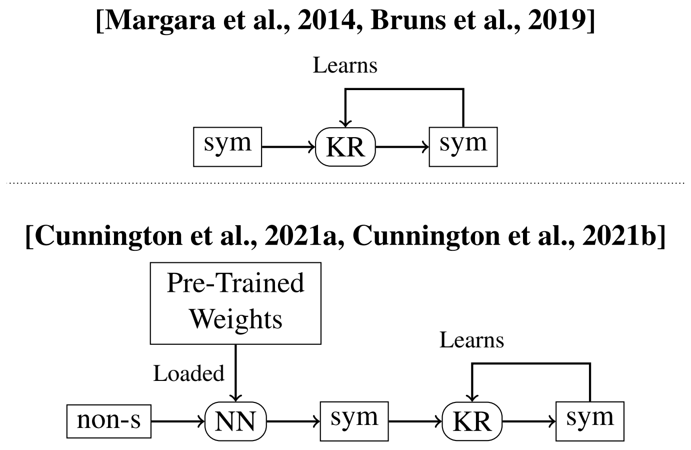

[Margara et al., 2014, Bruns et al., 2019]
Learns
sym
KR
sym
[Cunnington et al., 2021a, Cunnington et al., 2021b]
Pre-Trained Weights
Loaded
Learns
non-s
NN
sym
KR
sym

Figure 3.21: General schematic of the approaches that could be used to automatically learn complex event rules. Rectangular boxes represent data, using *sym* for symbolic data and *non-s* for non-symbolic data. Oval boxes represent algorithms, using *NN* for neural network components and *KR* for knowledge reasoning (symbolic) components.

While further research on the approaches described above would be useful for situations in which we do not know the complex event rules, for the rest of this thesis we have made the assumption that experts are able to provide us with such definitions. As a result, we explore the second option, which aims to train the neural network instead. As such, these approaches train the neural network to classify the non-symbolic input into symbolic information that is useful for the complex events we want to detect. In particular, the rest of the chapters explore how the human knowledge provided in the form of manually defined rules can be used to reduce the amount of training data required to train a CEP system. This makes it easier to obtain and label enough cases even in situations with sparse data. On top of that, we also explore how noisily labelled data can affect the performance of a system. In complex events, noisy labels can be particularly common, as different people may label the same situation differently. For instance, looking at the case of UCF-Crime, not everyone may be able to easily differentiate between the classes of burglary, robbery, stealing and shoplifting. This is because all of them may include an individual or a group taking something that does not belong to them. While the specific legal definitions for each of them may be different, it is not unreasonable to assume that the people assigning the labels might use the incorrect one, unless they are provided with very specific instructions about what constitutes each of them. Furthermore, there are also cases where more than one label could be applied. For instance, Abuse001 was labelled as abuse, but the perpetrators of this abuse also stole something. Similarly, Fighting003 shows a failed stealing attempt, which is stopped through a fight. Another video in UCF-Crime (Arrest004) shows the police arresting someone in a car, who crashes their car multiple times in an attempt to escape. While all of these are assigned a single label (abuse, fighting and arrest respectively), you could make an argument that other labels also apply to each of them. This argument could even be made more strongly for videos such as Arrest015, where two police cars crash into a van, presumably in an attempt to immobilize it to arrest the driver and/or passengers. In the video, however, the van manages to drive away. As such, while the label assigned in the dataset (arrest) makes sense, as it does seem like the police are trying to perform an arrest, the video does not show anyone being successfully arrested. As a result, it could be argued that the video should be labelled as a road accident instead, as there are vehicles crashing into each other but no actual arrest. This is further complicated due to the fact that the short descriptions provided by the creators for each label have some amount of overlap, in our opinion. Therefore, even with those descriptions, it seems likely that someone may classify some of the videos differently than the labels assigned in the dataset.

As such, it would be difficult to evaluate a system's robustness to noise using a dataset that comes from the real world. This is because, when evaluating this robustness, we want to have perfect control over how much noise there is in the data. This would be almost impossible using real-world data, as it would be very difficult to quantify the amount of noise that is already in the dataset. We have a similar issue when evaluating the performance of a system depending on the amount of data that it has used for training. For this purpose, we want to have a large dataset, which we can easily divide to create smaller datasets with only a percentage of the data. Due to its weak labelling of the entire video, indicating the type of abnormality but not when it happens, this is difficult to do using UCF-Crime. It is for these reasons that, for the rest of the thesis, we will be focusing on synthetically generated datasets, which offer us a much finer control of the size of the dataset, as well as its levels of noise. However, as part of these synthetic datasets we still require some non-symbolic data as the input. Since most non-symbolic datasets contain real-world data, this can be an issue, as we are again potentially introducing noise. This is part of the reason why we chose to use MNIST [LeCun et al., 2010] as the basis for our synthetic datasets, as it contains non-symbolic data but minimum to no noise.

Chapter 4

Training CEP Approaches Using Sparse Data

In this chapter, we focus on comparing different types of CEP approaches on their *data-efficiency* when training to identify complex events from a non-symbolic data stream. As such, the main focus of this chapter is on the *efficiency* of the approaches, particularly in terms of how much training data is required to train them.

For this purpose, we present two neuro-symbolic approaches, DeepProbCEP and Neuroplex. These approaches are able to train in an end-to-end manner to recognize complex events, meaning that they are able to learn from the subsymbolic input data using labels for only the complex events, without requiring the simple events to be labelled. Both approaches are designed to allow users to inject their human knowledge into the system, which simplifies the training problem and allows the systems to learn with far fewer data.

DeepProbCEP and Neuroplex were developed in parallel and in collaboration with the University of California, Los Angeles. These systems were designed to solve similar problems using different neuro-symbolic approaches. As such, there was continuous communication between the two sides with the intention of sharing our findings, which resulted in multiple shared publications [Roig Vilamala et al., 2020, Roig Vilamala et al., 2021, Xing et al., 2019, Xing et al., 2020]. As part of this communication, there was a discussion between both parts on how to evaluate the approaches. DeepProbCEP was mainly developed in Cardiff. Neuroplex was mostly implemented at University of California, Los Angeles, although we collaborated in the design of the logical layer.

The main difference between these approaches and ProbCEP is in the fact that ProbCEP relied entirely on proxy models, whereas DeepProbCEP and Neuroplex aim to learn how to detect complex events using training data. In particular, both DeepProbCEP and Neuroplex train the neural part of their architectures to classify simple events, which can then be used to detect complex events through the rule definitions from the logic layer. In this sense, the process of inference is quite similar between the three approaches. However, the process through which each approach gets to that stage is quite different.

After introducing DeepProbCEP and Neuroplex, we compare them with the neural-only approaches presented in Chapter 2. More specifically, we compare them with the LSTM and Convolutional 3D approaches in their performance after training with sparse data. Our hypothesis is that the neuro-symbolic approaches can learn to detect complex events with significantly less data, thanks to the injection of human knowledge into the system. In order to test this, we generate training datasets of varying sizes for the same problem, and compare how the accuracies change after training each of the approaches with the different dataset sizes.

The approaches compared in this chapter perform end-to-end training, meaning that they are only provided with training labels on when complex events are happening given an input stream of data. As such, ProbCEP cannot be fairly compared to the other approaches evaluated in this chapter in terms of accuracy, as it solves a different (albeit similar) problem.

4.1 Problem Definition

For this chapter, we will be considering a different problem definition for how complex events can be specified. This is because the approaches against which we want to compare DeepProbCEP do not support the Event Calculus based approach used in the previous chapter. Instead, they detect complex events when a pre-defined pattern of complex events happens within a given window. As a result, in order to be able to compare the approaches in a fair manner, we are evaluating all approaches using this type of complex event definition. It is worth noting, however, that DeepProbCEP is designed to support both problem definitions, as we demonstrate in Chapter 5.

As before, let $\mathcal{L}$ be a first-order language with three sorts, sets of many predicates, $\mathbb{S}$ for simple events, $\mathbb{C}$ for complex events and $\mathbb{T}$ timestamps. Timestamps must be positive integers or zero.

For this chapter, we assume that the function between $\mathbb{S}$ and $\mathbb{T}$ is bijective $\mathbb{S} \leftrightarrow \mathbb{T}$, meaning that for each timestamp there is exactly one simple event. This is due to the limitations of approaches from the literature against which we want to compare ourselves to. The predicate $happensAt(\mathbb{S}, \mathbb{T})$ can be used to define this relationship.

Complex events may occur at different points in time if a given combination of simple events occurs. For each type of complex event $\mathbb{C}$, a pre-defined pattern $P_{\mathbb{C}}$ of simple events must happen within a window of timestamps W. W must be a positive integer that defines the size of the sliding window used. W must be defined for the program as a whole. The pattern $P_{\mathbb{C}}$ defines the pattern for the complex event $\mathbb{C}$ and must consist of a non-empty sequence of simple events, which can be of any length smaller or equal to W. The predicate $holdsAt(\mathbb{C}, \mathbb{T})$ can be used to indicate that a complex event $\mathbb{C}$ occurs at a certain timestamp $t \in \mathbb{T}$. This will happen if the sequence of simple events $P_{\mathbb{C}}$ happens in the given order between t (included) and $t - W$ (excluded). Furthermore, in order to avoid multiple detections of a complex event from the same set of simple events, $holdsAt(\mathbb{C}, t)$ will only be true if the last element of $P_{\mathbb{C}}$ happens at timestamp t.

Example 1. Using an input stream with the sounds recognized in an audio recording, we could define the types of simple events to be:

$$S = \{siren, gunshot, car, \dots\} \tag{4.1}$$

Then, we can define the complex event $warning$ to be the sound of a siren followed by gunshots. This can be modelled with $P_{warning} = [siren, gunshot, gunshot]$. Note that there are multiple instances of $gunshot$ in the sequence, and thus the complex event will only hold after the sound of a siren is followed by multiple gunshots. This is, of course, optional and could be adapted to suit the needs of the user.

For example, using a $W = 4$, this is when the predicate $holdsAt(warning, t)$ would be true:

- *siren, siren, gunshot, car*: The complex event $warning$ does not hold at any point, as there are not enough instances of *gunshot*.

- *siren, car, car, gunshot, gunshot*: Again, $warning$ does not hold at any point, as there are too many events between the last gunshot and the siren. This is because, with $W = 4$, only the 4 last events are considered at any point.
- *siren, car, gunshot, gunshot, car*: In this case $holdsAt(warning, 3)$ is true.

4.2 DeepProbCEP

In this section, we explore DeepProbCEP, our neuro-symbolic approach to CEP. It allows us to inject human knowledge into the system, which facilitates the training process. This makes it possible to train the system with much smaller amounts of data than neural-only approaches. DeepProbCEP divides the task into two distinct levels; (i) *low-level perception*, where non-symbolic data is classified into the correct type of simple event using a neural architecture (implemented using a *perception layer*) and (ii) *high-level reasoning*, where complex events can be detected based on the rules that define them (implemented using a *reasoning layer*).

In DeepProbCEP, the perception layer is implemented by a neural network, which classifies the non-symbolic data into a pre-defined set of classes. This allows the system to extract the symbolic information from the input data, which can then be processed by the reasoning layer. This high-level reasoning consists of a probabilistic logic programming layer, where the user-defined rules are contained. This reasoning layer is responsible for detecting the situations under which the complex events occur, as defined by the user. In the following sections, we explain how both levels have been implemented.

This architecture is very similar to the ProbCEP approach, as it also combines a neural network with rule definitions. However, the main difference is in the fact that ProbCEP was designed to use pre-trained neural networks, while DeepProbCEP is designed to train them in an end-to-end manner.

4.2.1 Reasoning Layer

DeepProbCEP builds on top of DeepProbLog [Manhaeve et al., 2021] (also see Section 2.4.2). This allows us to define when the complex events happen using ProbLog rules, in a similar manner to ProbCEP. Using ProbLog code, users can define under which conditions each of the complex events they are interested in may happen. This can be done by defining a clause for each type of complex event, which needs to be true for the complex event to happen. As such, users need to define rules that specify under which conditions those clauses will be true, which will result in the complex event being active at that time.

In order to define these rules, users need to be able to evaluate when simple events are happening. The way to evaluate the state of simple events depends on their origin. If the simple events come from a symbolic input, these can be directly added to the reasoning layer without the need of going through the perception layer. This can be done by appending the clauses representing their values to the ProbLog code, for example. If the source of the simple events is non-symbolic data, this will need to be processed by the perception layer first. This can be done by using a neural AD, which allows communication between the perception and reasoning layers. For example, this can be done by adding the following line to the ProbLog code:

```
nn(mnist_net,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).
```

This defines the clause `digit(X, Y)`, which provides the argument `X` to the neural network `mnist_net`. This neural network has 10 different outputs, which we match to the digits between 0 and 9 using the list. Once this line has been added, the clause `digit(X, Y)` can be used to consult the perception layer. For this example, the value of `X` must be an MNIST image. For each value of `X`, and for each possible digit in `Y`, the clause will be true with a certain probability. This probability is determined by the output of the neural network in the perception layer using `X` as an input.

Once the simple events have been defined in the logic layer, the user needs to create rules that identify when the complex events occur based on the simple events. For this purpose, different frameworks can be used. Chapter 5 further explores how different frameworks give flexibility to DeepProbCEP, allowing users to define the rules for the complex events in their preferred manner. For this chapter, however, we only consider complex events defined as a pattern of simple events that must happen in order within a given window. This is done to allow us to compare DeepProbCEP with the relevant literature, which are restricted to this type of rule definitions. For DeepProbCEP, we define these patterns using some of the functionalities from the *sequence framework* (further explored in Section 5.2.1).

4.2.2 Perception Layer

The other part of the system is the perception layer, which is responsible for transforming the non-symbolic data into information that can be used to define rules in the reasoning layer. This transformation is performed by a neural network, which will classify the non-symbolic input data into a pre-defined set of classes. The structure of this neural network can be defined using PyTorch [Paszke et al., 2017]. This allows the user to make use of the most appropriate neural network architecture for classifying the input data being used. As such, as long as the appropriate neural network architecture is used any type of non-symbolic data can be used. There are, however, some restrictions on the implementation of the neural network, that come from the use of DeepProbLog to integrate the perception and reasoning layers. Currently, regression outputs are not supported by DeepProbLog. As such, only neural networks that classify into a set of classes can be used. Furthermore, the predicted values for these classes will be converted into an AC, with the score predicted for each class being treated as the probability for that class. This means that the probabilities for all the classes must add up to at most 1. This is usually archived by adding a SoftMax layer at the end of the neural network.

Once the reasoning and perception layers are defined, DeepProbLog [Manhaeve et al., 2018, Manhaeve et al., 2021] is used to train the perception layer neural network in an end-to-end manner.

It is worth noting that, after the training is performed, this perception layer neural network can be extracted. This can provide us with a neural network capable of classifying simple events. This can be a useful byproduct as we are assuming that we do not have access to the training data required to train such a neural network directly.

4.2.3 DeepProbCEP Setup Used in Experimentation

For the experiments performed below, we consider a stream of MNIST digits as the input to each of the systems. Complex events are defined as two instances of the same digit within a

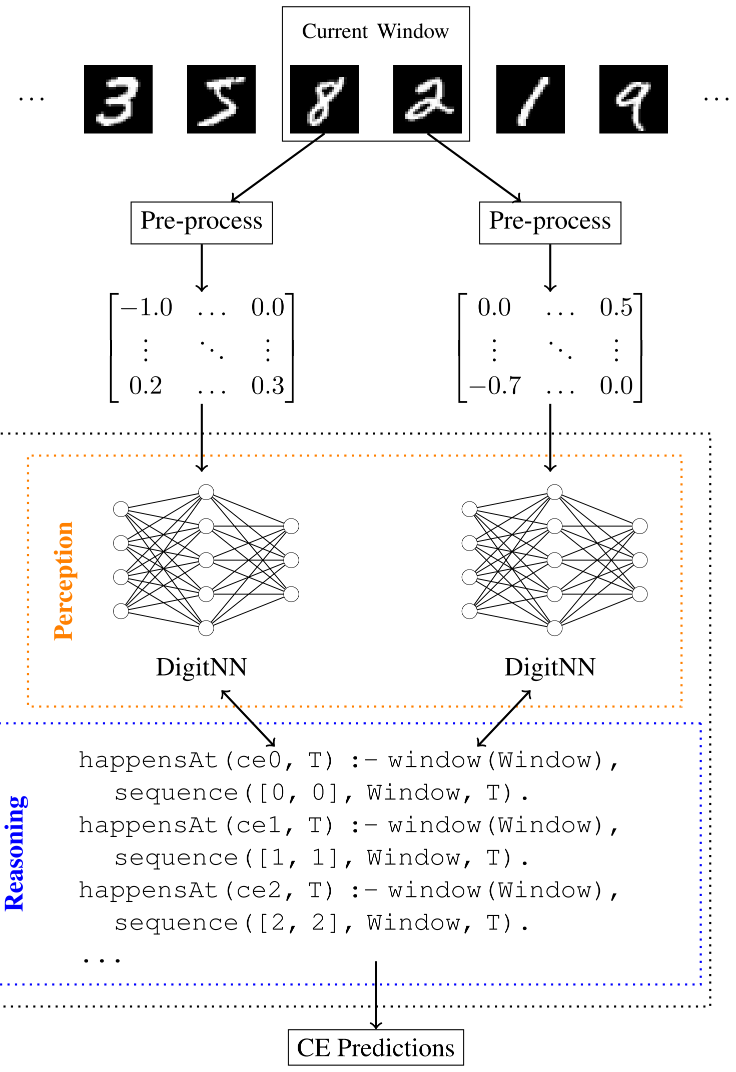

Current Window
...
...
Pre-process
Pre-process
DeepProbCEP
Perception
DigitNN
DigitNN
Reasoning
happensAt(ce0, T) :- window(Window),
  sequence([0, 0], Window, T).
happensAt(ce1, T) :- window(Window),
  sequence([1, 1], Window, T).
happensAt(ce2, T) :- window(Window),
  sequence([2, 2], Window, T).
...
CE Predictions

Figure 4.1: Overall architecture of DeepProbCEP for the MNIST setting. For details on the architecture of DigitNN, see Table 4.1.

given window, which will be specified for each experiment. One of these instances must occur at the last position in the window size, and the other can occur in any other position inside the window. If a window of simple events ends with a 0 and there is another 0 somewhere within the window, that will create $ce0$. If the last simple event is a 1 and there is another 1 within the window, that will create $ce1$, and so on. More details on how the datasets used in experimentation have been generated are given in Section 4.5.

Figure 4.1 shows how the DeepProbCEP approach can be applied to create a system capable of detecting such complex events. As shown in the figure, the first step is to pre-process the input

Table 4.1: Architecture for DigitNN, the neural network used to classify MNIST digits in the DeepProbCEP approach. The architecture of DigitNN is the same as the one used in the paper presenting DeepProbLog [Manhaeve et al., 2021], but the weights are randomly initiated and the neural network is trained using complex event training data.

Layer Type	Output Channels	Kernel Size	Stride
Conv2D	6	(5, 5)	(1, 1)
MaxPool2D	6	(2, 2)	(2, 2)
ReLU	6	-	-
Conv2D	16	(5, 5)	(1, 1)
MaxPool2D	16	(2, 2)	(2, 2)
ReLU	16	-	-
Reshape	256	-	-
Linear	120	-	-
ReLU	120	-	-
Linear	84	-	-
ReLU	84	-	-
Linear	10	-	-
Softmax	10	-	-

data. For the case of MNIST digits, we simply normalize the value of each pixel with a mean of 0.5 and a standard deviation of 0.5. Then, the matrix resulting from the pre-processing is fed into the perception neural network, DigitNN in Figure 4.1. This neural network identifies the different types of simple events from the input stream. In the case of DigitNN, it detects which digit appears in the image, returning a likelihood value for each of the 10 possible digits (0 to 9). The architecture of DigitNN (shown in Table 4.1) was proposed by the authors of DeepProbLog in [Manhaeve et al., 2021], where they explain it is based on the PyTorch tutorial. We chose to use the same architecture, as it had already been shown to work well with DeepProbLog's training approach. However, the weights for DigitNN are randomly initialized, meaning that there is no pre-training based on any other problem. Since the task of DigitNN is the same in both our setting and in [Manhaeve et al., 2021] (classifying MNIST images into the digit appearing on them), we can assume that the same architecture will perform well in both settings. However, thanks to the fact that different rules are defined on the reasoning layer, the tasks performed by the system as a whole are very different. As shown in Table 4.1, DigitNN consists of 2 sets of convolutional layers combined with a MaxPool layer and a ReLU activation function. After these, there are 3 fully connected layers. The first two layers have a ReLU activation function, whereas the last layer uses a Softmax function.

Finally, the prediction from the neural network is then used in the logic layer, which allows it to predict whether or not a certain complex event is happening at a certain point in time. This is based on the rules defined by the user, and any frameworks being used. A snippet of the code used in experimentation can also be seen towards the bottom of Figure 4.1. This shows how complex events $ce0$, $ce1$ and $ce2$ are defined, according to the description for each of the complex events given above. Namely, if a 0 happens at the end of the window and another 0 appears in any other position within the window size, this creates a complex event $ce0$. $ce1$ detects the same pattern using 1 as the digit, and so on for each of the 10 digits.

When using DeepProbCEP for experimentation, we set a maximum number of epochs of 100. However, in order to avoid overfitting, we also make use of early stopping with patience of 10 epochs (both values empirically determined). This means that if the performance of the system on the validation dataset does not improve for 10 epochs, we end the training early. We will then use the weights that performed the best in the validation dataset for testing. We also use the same optimization parameters for DigitNN as used in [Manhaeve et al., 2021]. More specifically, an Adam optimizer with a learning rate of 0.001.

4.3 Neuroplex

Neuroplex [Xing et al., 2020] is also a neuro-symbolic approach that makes use of human knowledge in order to reduce the amount of training data required. Neuroplex also makes use of two layers; the low-level perception layer and the high-level reasoning layer. However, the main difference between both approaches is the fact that Neuroplex implements the reasoning layer in a different manner than DeepProbCEP.

Neuroplex was developed by *University of California, Los Angeles* in collaboration with *Cardiff University*, with the aim of improving DeepCEP [Xing et al., 2019]. DeepCEP is an approach that uses a specifically designed CEP language to allow for the definition of complex events based on a non-symbolic input. Similarly to ProbCEP, DeepCEP combines pre-trained neural networks with user-defined rules to detect complex events. Through the use of a specifically designed CEP language, DeepCEP focuses on offering flexible rule definitions for complex events. Neuroplex was intended to improve on this by allowing the user to train the perception

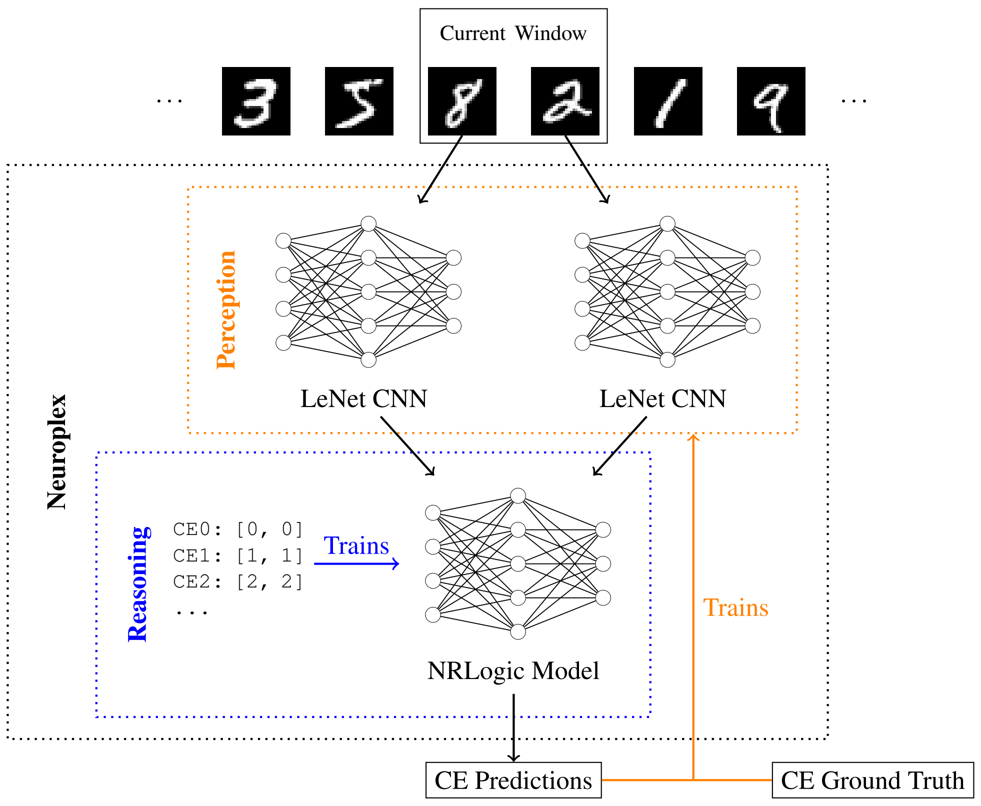


Figure 4.2: Overall architecture for Neuroplex.

neural networks, instead of relying solely on pre-trained models. However, at the time of writing, a system integrating the flexible rule definition from DeepCEP with the end-to-end training from Neuroplex has yet to be implemented. Instead, in its current implementation, Neuroplex only supports sequences of simple events —such as the ones considered in our experiments— for the definitions of complex events. As we want to compare Neuroplex against other approaches in our experiments, for the rest of the thesis we focus on the current implementation of Neuroplex[1], discussing its limitations and how they could theoretically be removed or minimized through an integration with DeepCEP.

Figure 4.2 shows the architecture used for the experiments shown in this chapter, based on the current implementation of Neuroplex approach. In Neuroplex, after the user has defined the rules for the complex events, a neural network is trained to emulate the behaviour of a logic layer that recognizes those rules (*NRLogic model* in the diagram). This is done by generating

[1]The latest implementation, at the time of writing (26th of July 2022), was published on the 25th of November 2020 and can be found at `https://github.com/nesl/Neuroplex/tree/938221da14`

a large number of possible inputs and training the neural network to output the correct label according to the logic rules. This process is called knowledge distilling [Hinton et al., 2015, Hu et al., 2016], and it can be used for a multitude of applications. In this case, it is used to transform the rules for the complex events into a neural network, thus making them differentiable. After this, the resulting neural network can be used as the reasoning layer. This NRLogic model is currently implemented using Long-Short Term Memory cells (LSTM). If the support for more flexible definitions of complex events was implemented, a more sophisticated architecture may be needed to accurately represent the more complex rules.

After distilling the knowledge into the NRLogic model, this NRLogic model is combined with the low-level perception neural network. In Neuroplex, this low-level perception is implemented using a Convolutional Neural Network (CNN). Before continuing the training, the NRLogic model is then frozen, meaning that the weights for the reasoning layer will not be modified by further training. This is done to avoid modifying the behaviour of the NRLogic model, as if that happened it would no longer accurately represent the behaviour of the manually defined rules. Finally, the system is trained in an end-to-end manner. This trains the low-level perception neural network to recognize the simple events into the classes used to define the complex events in the reasoning layer. This end-to-end training is made possible by the fact that the high-level reasoning is differentiable, which is why a neural network that emulates the rules needed to be used. Furthermore, the injection of human knowledge to train the NRLogic model allows Neuroplex to train with significantly less data than neural-only approaches [Xing et al., 2020].

	DeepProbCEP	**Neuroplex**
Training Speed	Slow	Fast
Inference Speed	Slow	Fast
Allows modifications of logic rules	Yes	Requires re-training
Flexibility of rule definitions	ProbLog [Fierens et al., 2015]	Sequence detection
Reliability of the rules inference	Logically sound	Potentially inconsistent

Table 4.2: Differences between DeepProbCEP and Neuroplex.

Both Neuroplex and DeepProbCEP have certain advantages and disadvantages when compared to each other, which are summarized in Table 4.2. DeepProbCEP's limitations mostly come from the use of an actual logic layer, which is significantly more time consuming to calculate than the neural network approximation used by Neuroplex. This means that Neuroplex is faster in both training and inference times. However, using a neural network to emulate the logic layer also causes a number of limitations for Neuroplex. Firstly, it means that every time the rules for the complex events are modified the system needs to be trained again. This is because the high-level neural network will need to be updated to reflect those changes. Secondly, the ways in which complex events can be defined are, in the current implementation, substantially more limited than in declarative approaches. As explained above, Neuroplex is only able to define rules as sequences of simple events.

Finally, while Neuroplex can generate synthetic data to distill the rules into the neural network, it is not possible for the user to know that the neural network will behave exactly as the rules define in all situations. This is due to the nature of the neural network, which may give an unexpected answer, particularly if the input has not been seen in the data generated for the knowledge distillation. The only way to guarantee that the neural network will always behave as expected is to evaluate every possible situation, which becomes unfeasible as the complexity of the problem increases. As a result, there is a risk that Neuroplex will not behave as expected in some situations.

If Neuroplex was integrated with DeepCEP, the flexibility of rule definitions would significantly improve. However, there would most likely still be difficulties ensuring that the NRLogic model accurately emulates the logic layer, particularly when the complexity of this logic layer increases. It may be possible to avoid this when performing inference, once the training is complete, if the logic layer is used directly, instead of being emulated. This should be possible as there is no need for the logic layer to be differentiable if no more training is required. While the direct use of the logic layer would most likely reduce the inference speed significantly, this would also allow users to change the complex event rules without requiring re-training. Despite this, the direct use of the logic layer during training is not possible in Neuroplex, as the reasoning layer needs to be differentiable, and thus the NRLogic model needs to be used. This means that, at least during training, there will always be a risk that Neuroplex's logic layer will behave in a manner inconsistent with the logic rules.

4.3.1 Neuroplex Setup Used in Our Experimentation

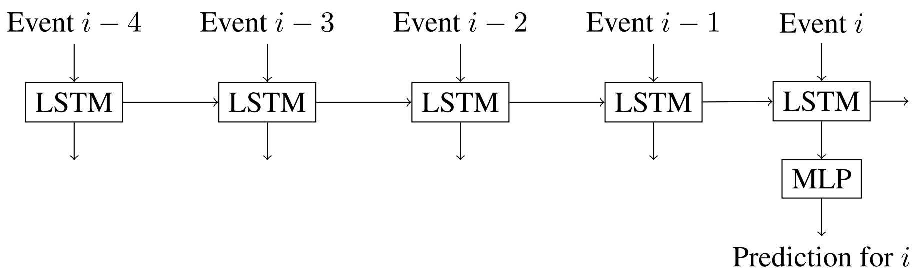
The problems solved by DeepProbCEP and Neuroplex are quite similar, but they have some small differences. In both problems, we are trying to detect certain pre-defined patterns that form complex events within a certain window. However, the Neuroplex system is designed to predict how many times each given pattern occurs in the given window. As such, Neuroplex outputs a float for each of the patterns that it is trying to detect. This float is then rounded to the nearest integer to generate the prediction that is shown to the user. For example, if a certain complex event occurs twice within the given window, Neuroplex should output a value close to 2.0. Then, the output will be rounded to 2 when shown to the user.

However, for the problem we are considering in our experiments, we are not concerned about how many times the pattern for the complex event happens. We are only concerned about whether it happens at least once or not. That is, we consider a boolean value for each of the complex events. In order to convert the output from Neuroplex to a type of output compatible with our type of problem, we simply evaluate any prediction of 1 or higher as a true value for that complex event happening. Predictions of 0 (after rounding) are considered as that complex event did not happen.

Consequently, for training, Neuroplex requires to know the number of times each complex event is happening. For all the other approaches evaluated in this chapter, we only provide the system with a label indicating which class is occurring in the given training window. However, Neuroplex is provided with the number of times each of the complex events is occurring (which will be 0 for all classes except the one that is occurring). This may give a slight advantage to Neuroplex, as it is provided with more information than the other approaches. However, preliminary experiments where we adapted the Neuroplex architecture to work with the problem definition used for DeepProbCEP significantly reduced its performance.

In our experimentation, we use a maximum number of epochs of 200. We also use early stopping with patience of 20. We change the rules used to match how the complex events are generated in our datasets. For all other parameters, we use the default values used in the original Neuroplex paper [Xing et al., 2020]. More specifically, we use LeNet [Lecun et al., 1998] as the perception layer and an LSTM network as the reasoning layer. LeNet is a convolutional

Figure 4.3: Architecture for the LSTM approach for a window of 5, evaluating for timestamp i. A smaller window would have fewer input events and a bigger window would have more. As can be seen in the figure, each hidden state is passed from each LSTM to the next. Finally, the output for the last LSTM is passed through a linear layer which outputs the prediction for the system.



neural network (CNN) with two convolution modules (convolution layer + Relu activation + maxpooling) with a 5-by-5 kernel, followed by two fully-connected layers. The reasoning layer consists of a single layer of LSTM with 64 hidden units, followed by a fully connected layer. Some experimentation was done on optimizing those parameters for this problem. However, other configurations showed no improvement.

4.4 Neural-only approaches

In this section, we explain how the neural-only approaches described above in Chapter 2 have been adapted to work with the datasets used for our experimentation.

4.4.1 LSTM Approach

For the LSTM approach, we combine an LSTM cell with a Multi Layer Perceptron (MLP). As shown in Figure 4.3, each of the simple events in the given window is fed into the LSTM cell. The order in which the events are fed into the LSTM cell is the same as the order in which they come in. The hidden state is passed through the LSTM cells and, finally, the prediction is based on the output of the last LSTM in the window. The structure of the neural network used for this approach can be seen in Table 4.3

When using the LSTM approach, we have set a maximum number of epochs of 500. However, we are also using early stopping with patience of 15. This means that, if performance on the

Table 4.3: Architecture for the LSTM approach. W is an input variable that represents the size of the window.

Layer Type	Input Size	Output Size
Reshape	W, 28, 28	W, 784
LSTM	W, 784	W, 100
Linear	100	11
Softmax	11	11

validation dataset does not improve for 15 epochs, we stop the training early. Then, the weights that gave the best performance on the validation dataset will be used to test the model. This is done to avoid overfitting the model. In all our experiments, this approach stopped before reaching 500 epochs.

4.4.2 C3D Approach

For the C3D approach, we used the architecture shown in Table 4.4. As can be seen in the table, the architecture consists of two pairs of a Conv3D layer plus a MaxPool3D layer. This is followed by a set of linear layers. Table 4.4 also shows that the dimension of the kernel size for the Convolutional 3D layers depends on the window size. This is because that is the time dimension, meaning that when we change the number of simple events we are considering we have to adapt the architecture to match that value. The values of $T1$ and $T2$ are defined to always have an output with the shape of 32 channels with a shape of (1, 4, 4).

Table 4.4: Architecture for the C3D approach. $T1$ and $T2$ are variables that change depending on the $Window$ used. More specifically, $T1 = ceil(Window/2)$ and $T2 = ceil((Window + 1)/2)$, where $ceil$ rounds up to the nearest integer.

Layer Type	Output Channels	Kernel Size	Stride
Conv3D	16	($T1$, 5, 5)	(1, 1, 1)
MaxPool3D	16	(1, 2, 2)	(1, 2, 2)
Conv3D	32	($T2$, 5, 5)	(1, 1, 1)
MaxPool3D	32	(1, 2, 2)	(1, 2, 2)
Linear	256	-	-
Linear	128	-	-
Linear	64	-	-
Linear	32	-	-
Linear	11	-	-

For this approach, we also use a maximum number of epochs of 500 and an early stopping with patience of 15. In our experiments, this approach stopped before reaching 500 epochs in all cases as well.

4.5 Datasets Generation

In this section, we explain how we generated the synthetic datasets used in Section 4.7 to evaluate how the different approaches perform when training with sparse data. This is archived by generating a base dataset and then using different amounts of the training samples for each class to generate the data-scarcity datasets. The process of generating this base dataset requires some labelled data, which will be used to represent the input stream of simple events. Based on the labels assigned to this data and a set of pre-defined conditions indicating how the simple events combine into complex events, we can generate the labels for the complex events. For the purpose of these experiments, we chose to use the MNIST dataset [LeCun et al., 2010] for this purpose. This was mainly because previous implementations of both Neuroplex and DeepProbLog had already used MNIST-based datasets, making it easier to make a fair comparison between the approaches. However, the process of generating the base datasets is designed to easily allow users to change which dataset they rely on, as well as which conditions should be considered, both of which we use later in the thesis (Sections 5.2.4 and 5.2.2, respectively).

For the experiments shown in this chapter, all the approaches we will be comparing make use of a sliding window to detect complex events. In order to generate a complex event, all of the simple events that will be aggregated need to be within this sliding window at the same time. As such, the size of the sliding window limits the maximum span of time for a complex event. The value for the size of the sliding window needs to be determined before the datasets are generated, as the training labels need to match that window size. It might seem best to use a window size as big as possible to detect as many complex events as possible. However, by increasing the window size we increase the number of simple events we are considering, which makes the problem of finding a pre-defined pattern harder. This is especially true if we want the system to be fast. Furthermore, sometimes we might want to limit the window size as, if two simple events are too distant from each other temporally, there might not be a relationship between them. As such, we will generate both the base datasets and the data-scarcity datasets for different sliding window sizes.

For all datasets, we are using the same definitions of complex events. Specifically, we are looking for patterns where the digit that appears in the last position of our sliding window is repeated at some other point within the window size. Each of the digits between 0 and 9 creates a different complex event. We will refer to each of the complex events with the form ceN, where N will be the digit that generates the complex event. For instance, $ce0$ would occur when a 0 appears as the last simple event in the window and somewhere else within the window.

4.5.1 Base Dataset

In this section, we describe how we generated the base dataset. This dataset will be the base from which other datasets can be generated, using the modifications described in the following sections. The process used to generate the base dataset allows us to change the window size by changing the value of $Window$. $Window$ is a positive integer that indicates the number of timestamps between the first and last simple events that form a complex event. For this thesis, we have generated datasets with window sizes of 2, 3, 4, 5, 10 and 15.

In order to generate the base dataset, we use images of handwritten digits from the MNIST dataset. We have split the MNIST dataset into training, validation and testing, with 50,000, 10,000 and 10,000 images, respectively. Each of these is used to generate the corresponding parts for the base datasets. Algorithm 9 shows the code to generate such datasets, and the procedure is illustrated in Figure 4.4. As shown in both of these, C is filled with complex event labels for each timestamp in the sequence S, which is a randomly shuffled version of the original dataset (in this case, MNIST). For each timestamp, C will have the label for the corresponding complex event (in this case, $ce0$, $ce1$, ...) or $null$, which represents the case where no complex event occurs. In Algorithm 9, we first create S by shuffling the original dataset and initialize the outputs TS and C to be arrays of the same length as S. Then, for each timestamp in S we save the image in TS, which will be the input for the training. Then, we determine the label of which complex event is happening at each timestamp (lines 6 to 9). By default, we assume that no complex event is happening, assigning $null$ to each timestamp (line 6). However, if the conditions for a complex event are met, we substitute this by the corresponding complex event (lines 7 to 9). In this case, we check all the timestamps in the window except for the last element of the window (line 7) and check if the digit in the image matches the one in the last element of the window (line 8). If that is the case, in line 9 we set the label for timestamp T to the corresponding complex event (of the form $ce0$, $ce1$, ... based on the value of the matching digit). If we wanted to change the rules for the complex events, we would need to modify the code between lines 7 and 9 so that they assign the label we want based on the new rules.

Algorithm 9 Base Dataset Generation

```
Input: $originalDataset$, a list containing the original MNIST dataset. For each element $i$ in
    the list, $originalDataset[i].image$ and $originalDataset[i].digit$ can be accessed, which
    represent the MNIST image and the integer value of the digit shown in that image, respect-
    ively. $Window$, which is the window size that should be used to generate the dataset.
Output: $\langle TS, C\rangle$, where $TS$ contains the input data for the training dataset—a list of MNIST
    images—, and $C$ is the list of labels for that training data—a list of which complex event
    is detected for each timestamp (or “null” if no complex event occurs).
 1: $S \leftarrow shuffle(originalDataset)$   ▷ $S$ is a randomly shuffled version of $originalDataset$
 2: $TS \leftarrow new\ Array[0, length(S) - 1]$
 3: $C \leftarrow new\ Array[0, length(S) - 1]$   ▷ $TS$ and $C$ are arrays of the same length as $S$
 4: for all $T \in [0, \ldots, length(S) - 1]$ do
 5:     $TS[T] \leftarrow S[T].image$   ▷ $TS$ is filled with the images from $S$ for each timestamp $T$
 6:     $C[T] \leftarrow$ “null”
 7:     for all $P \in [T - Window + 1, \ldots, T - 1]$ do
 8:         if $S[P].digit = S[T].digit$ then
 9:             $C[T] \leftarrow$ “ce” $+ S[T].digit$
10: return $\langle TS, C\rangle$
```

4.5.2 Data-Scarcity Datasets

In order to evaluate the performance of the different approaches after training with different amounts of training data, we have created the *data-scarcity datasets*. These datasets vary a lot in size, ranging from 100 to 400,000 training points[2]. These data-scarcity datasets have been generated by performing either undersampling or oversampling on the base datasets.

The use of oversampling is what allows us to generate datasets as big as 400,000 training points,

[2]A total of 10 different training dataset sizes have been used: 100, 500, 1,000, 2,500, 5,000, 10,000, 25,000, 50,000, 100,000, 400,000

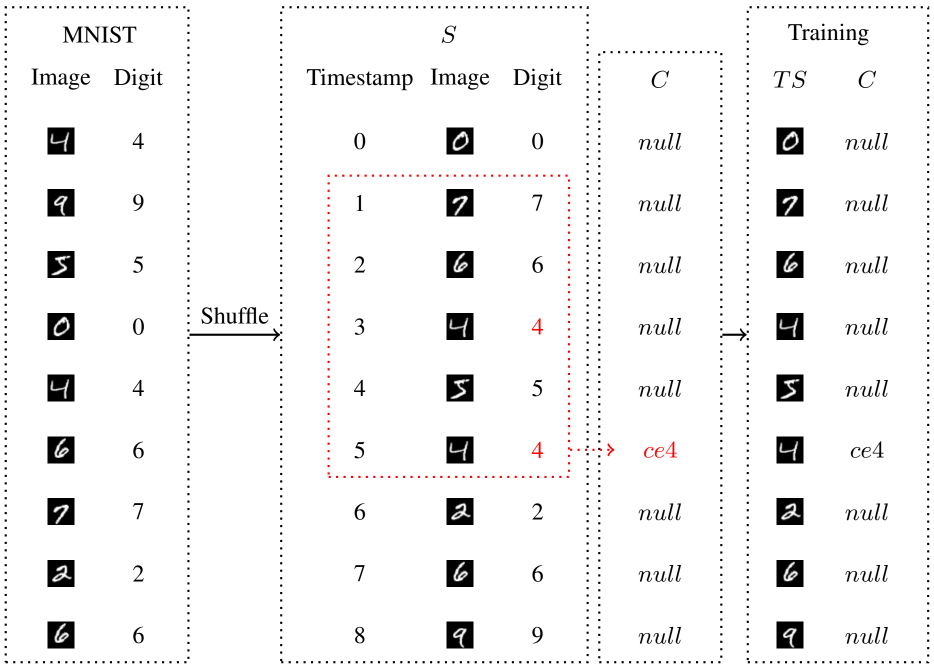


Figure 4.4: Diagram representing how the datasets used in this paper are generated. A window of 5 has been used. $ce4$ **is detected at timestamp 5.**

despite only having 50,000 initial images. The bigger datasets (50,000 to 400,000) use all of (or most of) the null cases generated in the base datasets and oversample the cases for the rest of the classes to create a balanced dataset. This is because there are many more cases of the null class in the base dataset, as it is much more likely for no complex event to happen than any other individual compex event, given our complex event definition. While this does increase the risk of overfitting, we wanted to test how the approaches would perform when training with as much data as possible. Furthermore, as shown in the results below, all approaches seem to either maintain or increase their accuracy on the testing dataset as the size of the training dataset increases. This seems to indicate that overfitting is not an issue in this case, as the middle sized datasets (5,000 to 25,000) do not use oversampling and thus should not increase the risk of overfitting. These middle sized datasets use all or most of the training samples for each of the complex event classes from the base datasets, and perform undersampling to randomly select the same number of cases for the null class. This is done to keep the datasets balanced. The smaller datasets (100 to 2,500) are generated by performing a balanced undersampling for the different classes. In this process, we randomly select only a few of the training examples from the base dataset in order to get to the total amount of training examples required. This does mean that the smaller datasets discard some training examples for all the classes, which is done to emulate how approaches would perform if we had less training data.

The different dataset sizes we have used, as well as their class distribution, can be seen in Table 4.5. As the table shows, the number of instances for each of the classes in a given dataset size differs by at most one, meaning there should be no overfitting due to a larger proportion of cases in the training dataset.

Table 4.5: Distribution of instances on the data-scarcity datasets. The distribution is the same for all window sizes.

Total	$null$	$ce0$	$ce1$	$ce2$	$ce3$	$ce4$	$ce5$	$ce6$	$ce7$	$ce8$	$ce9$
100	10	9	9	9	9	9	9	9	9	9	9
500	46	45	45	46	45	45	45	46	46	46	45
1000	91	90	91	91	91	91	91	91	91	91	91
2500	228	227	227	227	227	227	227	227	228	228	227
5000	455	454	454	455	454	454	455	455	455	455	454
10000	910	909	909	909	909	909	909	909	909	909	909
25000	2273	2272	2272	2273	2273	2272	2273	2273	2273	2273	2273
50000	4546	4545	4545	4546	4545	4545	4545	4546	4546	4546	4545
100000	9091	9090	9091	9091	9091	9091	9091	9091	9091	9091	9091
400000	36364	36363	36363	36364	36364	36363	36364	36364	36364	36364	36363

4.6 Experiment Design

In this section, we define the experiments performed on DeepProbCEP (Section 4.2), Neuroplex (Section 4.3), LSTM (Section 4.4.1) and C3D (Section 4.4.2).

In *Experiment 1: Performance with sparse data*, we aim to find how much each approach is affected by using small datasets for training. This will allow us to evaluate how well the objective of limiting the cost of obtaining training data is fulfilled. The data-scarcity datasets are used for this experiment. After training each approach with each of the dataset sizes, we compare their performance in terms of classification accuracy. Naturally, we would expect all approaches to perform worse as the size of the training dataset is reduced. However, we are interested in which approach is the least affected by this reduction of the training dataset size. The approach that is the least affected will be the most robust when training with sparse data. As complex events tend to be rare, this would be desirable, because training datasets will tend to be small in real-world scenarios.

In *Experiment 2: Performance on simple events*, we will focus on the accuracy when classifying simple events after training in an end-to-end manner. This is thanks to another advantage of the neuro-symbolic approaches, which allows them to learn how to classify simple events despite the fact that they have only been trained with information about the complex events. This is made possible by the introduction of human knowledge into the system, that logically describes the complex events as a function of simple events. Therefore, a byproduct of training a neuro-symbolic system is a simple event classifier. This could be quite useful in a real-world scenario, as we are assuming that we do not have the labels to directly train a simple event classifier. Our second experiment will evaluate the performance of DeepProbCEP and Neuroplex when classifying these simple events. For that purpose, we will also use the data-scarcity datasets for training, as for the previous experiment. This will also allow us to see how training with smaller datasets affects each of the approaches when classifying simple events.

For *Experiment 3: Time-efficiency*, we will compare the time-efficiency of DeepProbCEP and Neuroplex when training. As explained above in Section 4.3, Neuroplex's approach of emulating the logic layer has some drawbacks in terms of flexibility when defining rules and a risk that the neural network used for emulating might output a different result than the logic layer it is emulating. DeepProbCEP solves those issues by using an explicit logic layer. However, this does come at a cost in terms of time-efficiency, which makes DeepProbCEP slower than Neuroplex. In the third experiment, we will evaluate how significant the difference in terms of training time-efficiency is between the two approaches.

It is important to note that, in all cases, the approaches are tested with a dataset with the same window size as the dataset they were trained in.

It is also worth noting that DeepProbCEP does not use the training cases in which no complex event occurs. Instead, the training process for DeepProbCEP simply skips those cases. This is because, as a consequence of how DeepProbCEP works, there is no easy way to train it using cases in which no complex event occurs. In DeepProbCEP, we consider that no complex event is occurring if the probability for all the complex event types to be happening is very low. Since there is no rule defining the case in which none of the complex events happens, it is difficult to train the system using such a case. While future research could explore how to train in such case based on the fact that none of the other rules apply, we have opted for the simpler option of not using the cases where no complex event occurs when training. This, however, does not prevent DeepProbCEP from detecting such situations. Instead, DeepProbCEP is able to identify this by detecting when none of the rules for the complex events is activated.

The reason why we have generated training data for the case in which no complex event occurs is that the neural-only approaches require this type of training data. This is because these approaches would not be able to learn in which cases there is no complex event without training data including such situations.

Neuroplex has been shown to be able to train without requiring data for the cases in which the complex event does not occur [Xing et al., 2020]. This is, again, thanks to the injected human knowledge in the form of the rules for the complex events. However, thanks to the use of a neural network to emulate the logic layer, Neuroplex is also able to easily use training cases in which no complex event occurs, as the neural-only approaches. This means that Neuroplex is able to train in either of the two situations, whether the training data includes cases with no complex events or not. For the results shown in this thesis, Neuroplex has been trained using all the training samples in the dataset, including cases in which no complex event happens, as this seems to produce slightly better results.

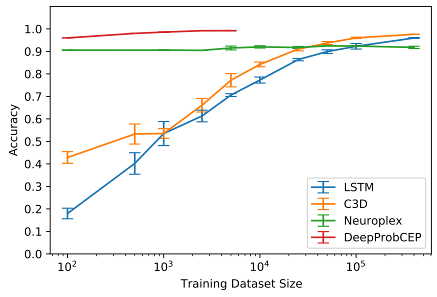
4.7 Experimental Results and Analysis

In this section, we explore the results of the experiments defined in Section 4.6. The following sections compare the performance of DeepProbCEP with the state-of-the-art approaches introduced in Chapter 2.

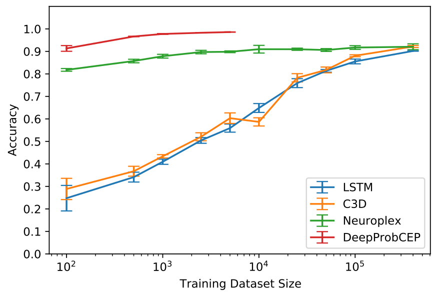
All the values displayed on the graphs and tables in the following sections are the result of averaging the accuracies of 3 different executions. The standard deviations of these accuracies are also shown in the graphs as error bars.

4.7.1 Performance with sparse data

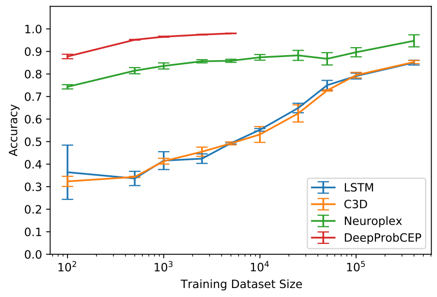
Figure 4.5 shows the classification accuracy for each approach in the different window sizes and after training on the data-scarcity datasets. As we can see, DeepProbCEP consistently



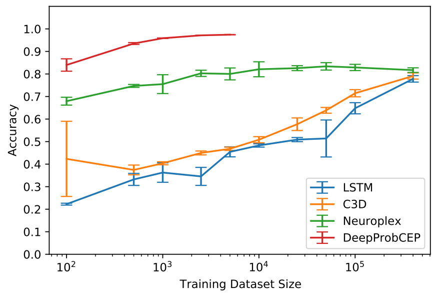
(a) Window of 2



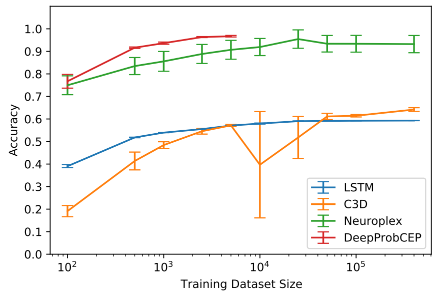
(b) Window of 3



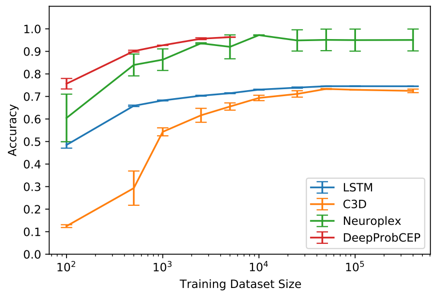
(c) Window of 4



(d) Window of 5



(e) Window of 10



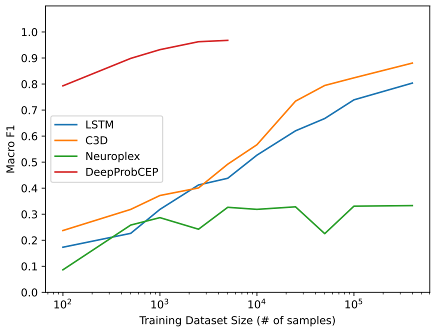
(f) Window of 15

Figure 4.5: Accuracy when detecting complex events for each of the considered approaches in different windows. Horizontal axis indicates the size of the training dataset used. Graphs show average results over 3 different executions, with the standard deviations shown as error bars.

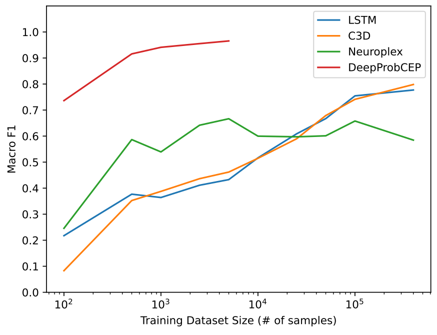
outperforms the neural-only approaches. The graphs only show results for DeepProbCEP with datasets of up to 5,000 training points. This is because DeepProbCEP takes a significantly longer time to train than the other approaches, which makes it difficult to train with large datasets. Despite this, training DeepProbCEP with just 100 data points can result in higher complex event classification accuracy than either of the neural-only approaches achieve when trained using 400,000 data points. This is despite the fact that we are using the best performing architecture we have been able to find through manual optimization for both the LSTM and C3D approaches, which is reflected by the performances of both approaches when trained with enough data. However, by its very nature, any neural-only approach used to solve this problem will need to learn to differentiate each of the simple event types at the same time that it is learning the rules that define the complex events. This is certainly possible given enough training data, as we can see in the graphs. However, in problems where we have sparse data, the performance will be severely impacted.

As explained above, the neuro-symbolic approaches get around this problem through the injection of human knowledge into the system. For this problem, the user must specify the rules that define a complex event, which allows the system to focus on learning how to classify the simple events. This is reflected in the performances obtained by both neuro-symbolic approaches, Neuroplex and DeepProbCEP. As we can see in Figure 4.5, both neuro-symbolic approaches classify complex events with an accuracy significantly higher than the neural-only approaches. We can also see that DeepProbCEP significantly outperforms Neuroplex, particularly when considering small window sizes, see Figures 4.5a, 4.5b, 4.5c, 4.5d (Window sizes of 2 to 5). Furthermore, even in the bigger window sizes where Neuroplex performs its best, DeepProbCEP gets slightly better and more consistent results, as shown in Figures 4.5e and 4.5f (Window sizes of 10 and 15, respectively). It is important to remark that, before the experiments in this paper, Neuroplex had almost exclusively been tested with window sizes of 10 and bigger. The only exceptions are two cases [Xing et al., 2020] that are intentionally designed to have significantly less complexity in order to allow the baseline models to learn. Therefore, some fine-tuning of the architecture might be necessary to get better performances in smaller window sizes. In contrast, the fact that DeepProbCEP directly uses logic programming for the logic layer means that it will always perform as expected, without requiring any fine-tuning.

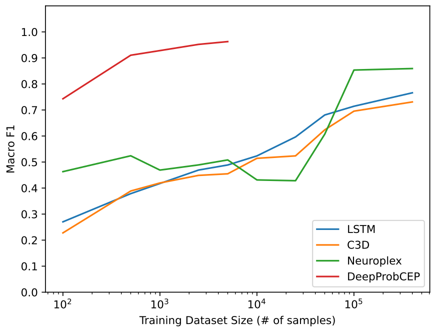
Looking at the macro F1 values, shown in Figure 4.6, we can again see that DeepProbCEP outperforms all other approaches. We have chosen the macro F1 metric to take into account the fact that the number of null classes in the test dataset is significantly higher, making it unbalanced. The neural-only approaches, which tend to have similar performances to each other, consistently have significantly lower values than DeepProbCEP. Meanwhile, for window sizes of 10 and 15 (Figures 4.6e and 4.6f), Neuroplex performs better than the neural-only approaches, albeit slightly worse than DeepProbCEP. However, the macro F1 values for Neuroplex in smaller window sizes (Figures 4.6a, 4.6b, 4.6c and 4.6d) is very unreliable, performing similarly or



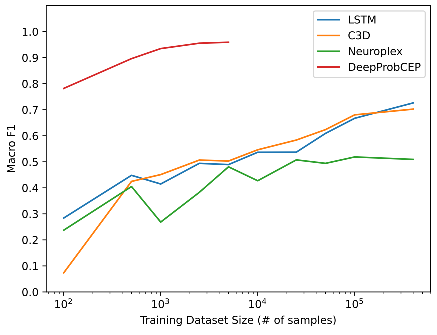
(a) Window of 2



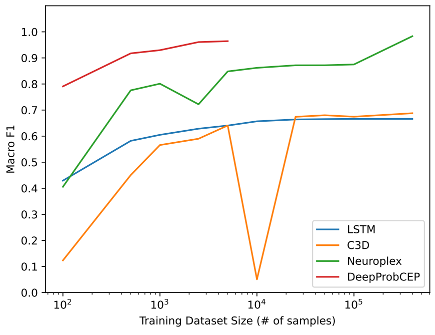
(b) Window of 3



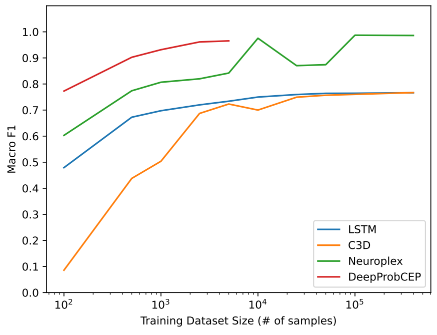
(c) Window of 4



(d) Window of 5



(e) Window of 10

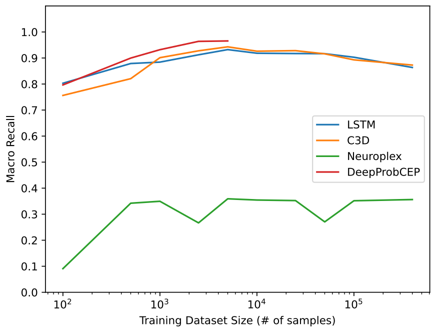


(f) Window of 15

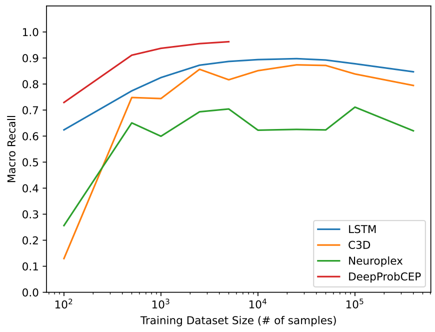
Figure 4.6: Macro F1 results when detecting complex events for each of the considered approaches in different windows. Horizontal axis indicates the size of the training dataset used.

even worse than the neural-only approaches in some cases.

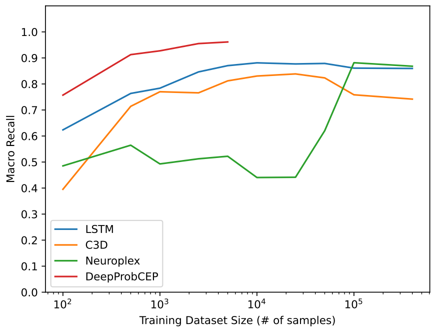
In terms of the macro recall (Figure 4.7), Neuroplex seems to perform even worse for window sizes between 2 and 5, being significantly outperformed by the neural-only approaches in many cases. Despite this, it does still outperform the neural-only approaches when using the bigger window sizes (Figures 4.7e and 4.7f). DeepProbCEP is still the best performing approach in general, although the neural-only approaches are much closer in recall results when using a



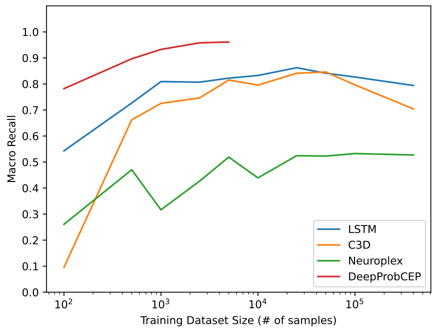
(a) Window of 2



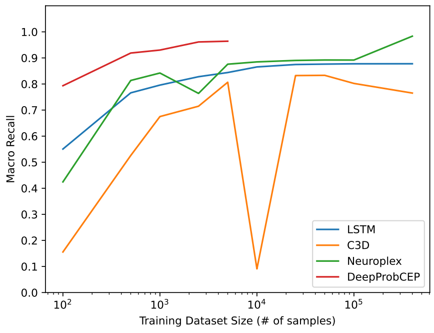
(b) Window of 3



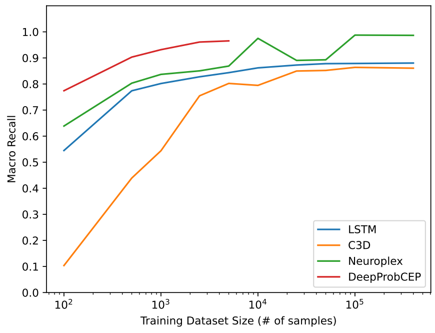
(c) Window of 4



(d) Window of 5



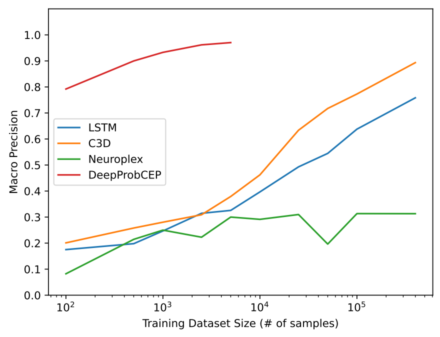
(e) Window of 10



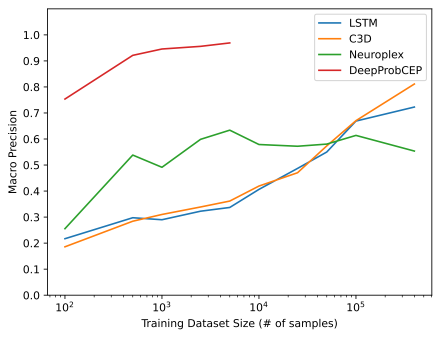
(f) Window of 15

Figure 4.7: Macro recall results when detecting complex events for each of the considered approaches in different windows. Horizontal axis indicates the size of the training dataset used.

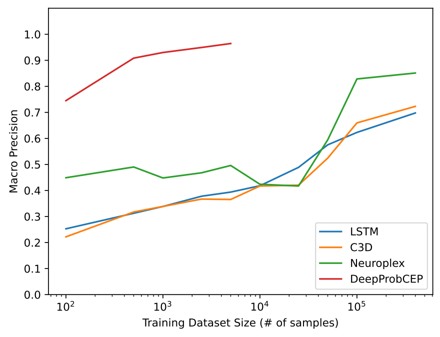
window of 2. In fact, the LSTM approach is ever so slightly better when using the dataset of size 100. Despite this, taking into consideration the difference in performance on terms of accuracy and F1 results, we would still recommend DeepProbCEP in all cases. Interestingly, the performance with macro recall seems to slightly decrease in some cases when training with the biggest datasets. This could be caused by either a slight overfitting or the fact that the accuracy is used to select the best performing model. As the accuracy is not weighted by the number of cases for each class, this could be causing the macro recall to perform slightly worse in some cases.



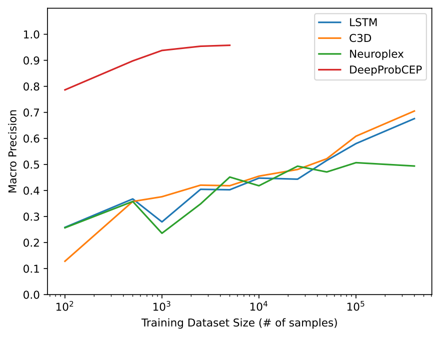
(a) Window of 2



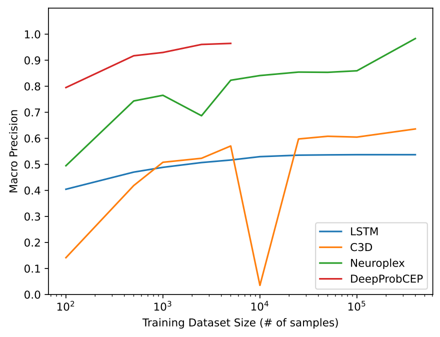
(b) Window of 3



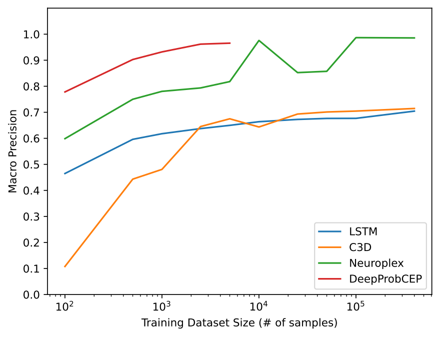
(c) Window of 4



(d) Window of 5



(e) Window of 10

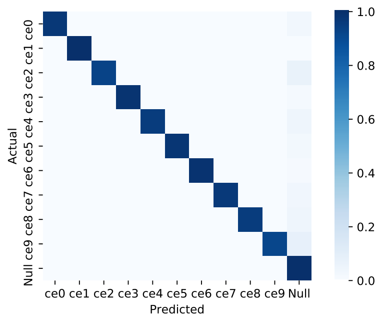


(f) Window of 15

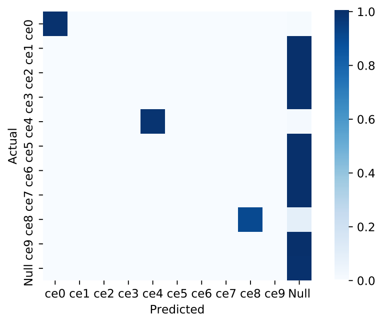
Figure 4.8: Macro F1 results when detecting complex events for each of the considered approaches in different windows. Horizontal axis indicates the size of the training dataset used.

Finally, we consider the macro precision (Figure 4.8). Again, DeepProbCEP has the best performance, with the only comparable approach being Neuroplex when using window sizes of 10 and 15, which still performs slightly worse. Taking all the results into consideration, DeepProbCEP seems like the most appropriate solution to this problem, as it is consistently the best performing and most reliable approach.

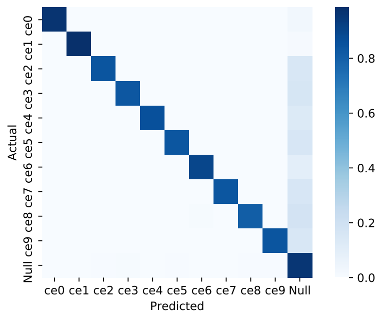
Types of error



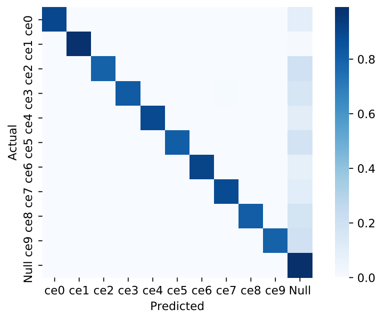
(a) DeepProbCEP



(b) Neuroplex



(c) LSTM

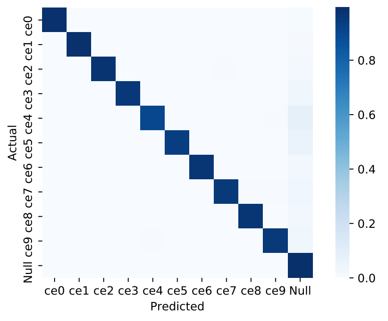


(d) C3D

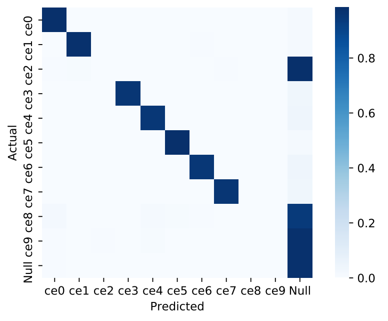
Figure 4.9: Confusion matrices for Window of 2.

In this section, we explore what types of error each of the approaches tends to make by looking at their confusion matrices. For all the confusion matrices shown in this section, the best performing training dataset size has been used for the corresponding approach and window size (which generally corresponds with the biggest dataset size, as shown above in Figure 4.5). These are shown in Figures 4.9, 4.10, 4.11, 4.12, 4.13 and 4.14, which show the results for each approach using a window of 2, 3, 4, 5, 10 and 15, respectively. The values have been normalized in the horizontal axis in order to allow comparison between classes that are less common than others in the test dataset.

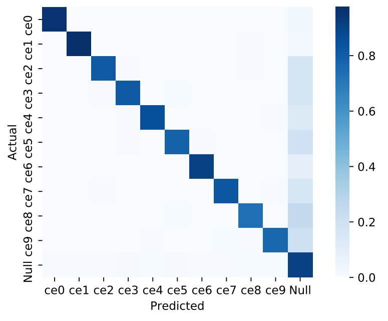
As shown in Figures 4.9b, 4.10b, 4.11b and 4.12b, Neuroplex tends to incorrectly classify some of the classes when using smaller window sizes. This matches our observations that Neuroplex does not perform as well with smaller window sizes. Particularly, it seems to misclassify some of the complex event classes as the null class. This, however, does not happen in bigger window sizes, where Neuroplex classifies most instances correctly (see Figures 4.13b and 4.14b).



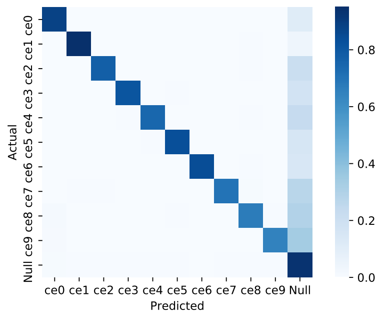
(a) DeepProbCEP



(b) Neuroplex

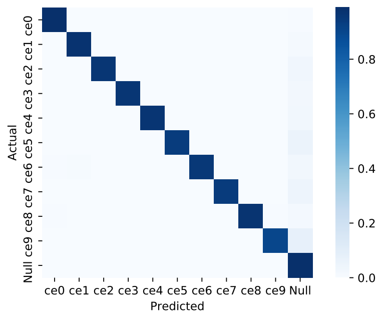


(c) LSTM

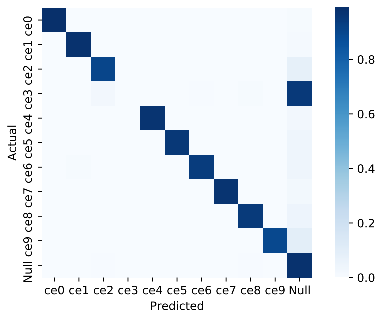


(d) C3D

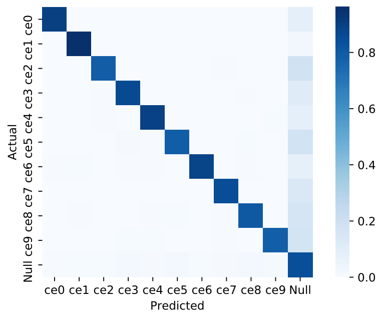
Figure 4.10: Confusion matrices for Window of 3.



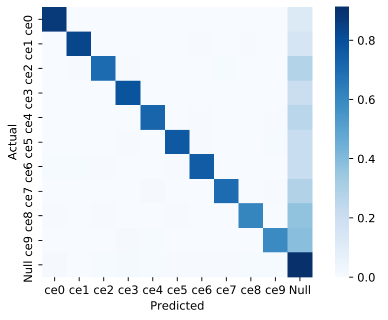
(a) DeepProbCEP



(b) Neuroplex

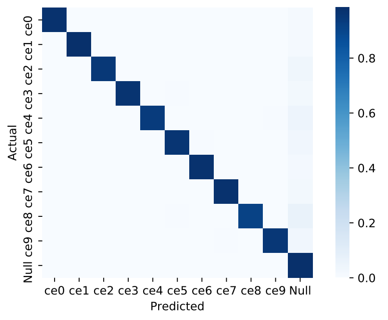


(c) LSTM

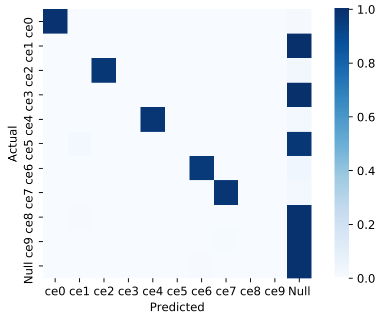


(d) C3D

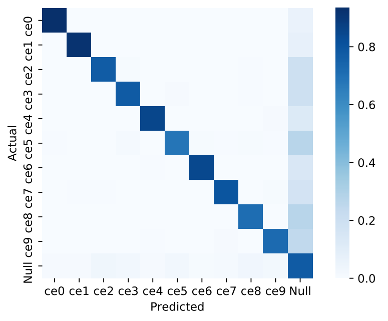
Figure 4.11: Confusion matrices for Window of 4.



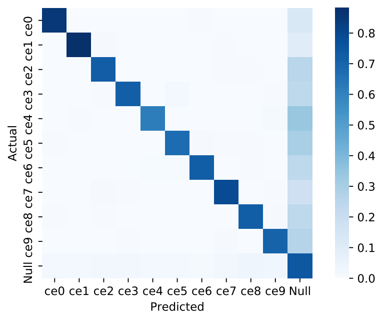
(a) DeepProbCEP



(b) Neuroplex

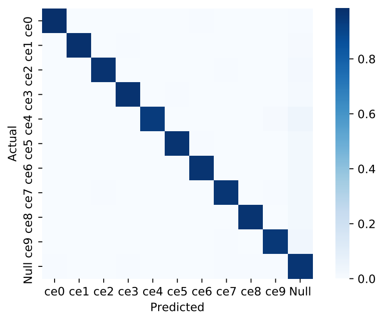


(c) LSTM

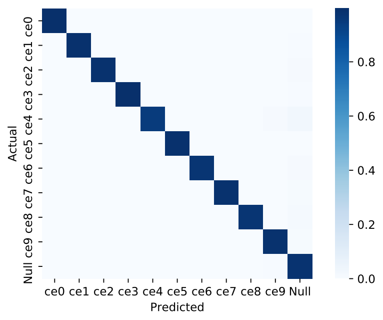


(d) C3D

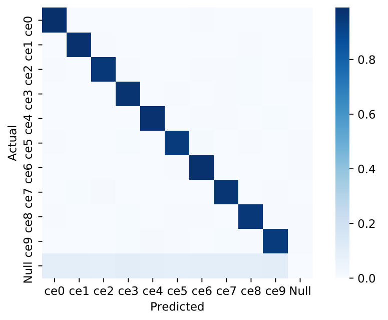
Figure 4.12: Confusion matrices for Window of 5.



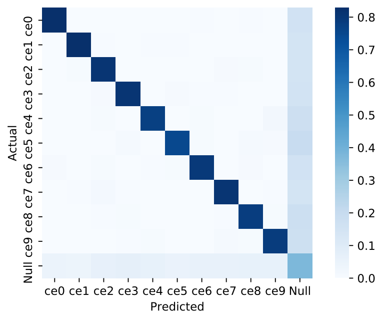
(a) DeepProbCEP



(b) Neuroplex

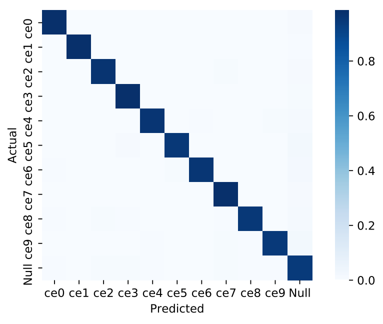


(c) LSTM

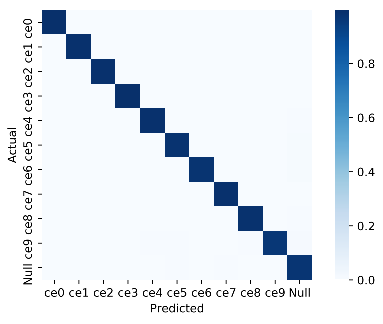


(d) C3D

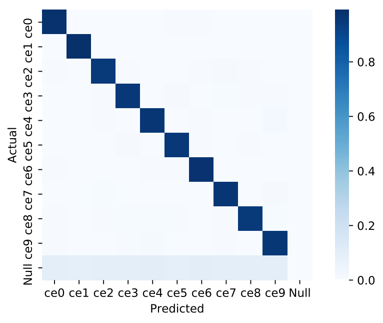
Figure 4.13: Confusion matrices for Window of 10.



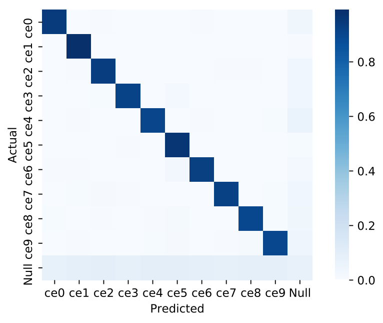
(a) DeepProbCEP



(b) Neuroplex



(c) LSTM



(d) C3D

Figure 4.14: Confusion matrices for Window of 15.

Similarly, the most common mistake in the neural-only approaches (LSTM and C3D) in smaller window sizes (between 2 and 5) is also classifying cases with a complex event as the null class. However, there is a clear difference between the approaches, as Neuroplex seems to consistently fail at classifying a few of the classes, but getting the rest of them correctly almost all of the time. In contrast, the neural-only approaches tend to incorrectly classify a small percentage of almost every complex event class as the null class.

The contrast between the neuro-symbolic and the neural-only approaches is even clearer in the bigger window sizes (10 and 15), as both neuro-symbolic approaches classify almost all of the cases correctly, as shown in Figures 4.13a, 4.13b, 4.14a, 4.14b. Meanwhile, Figures 4.13c, 4.13d, 4.14c, 4.14d show how the neural-only approaches, and particularly the LSTM approach, seem to perform fairly well at classifying the cases where a complex event is happening. However, both approaches perform quite badly when classifying a case that belongs to the null class, where we are trying to identify that none of the patterns that define a complex event is occurring.

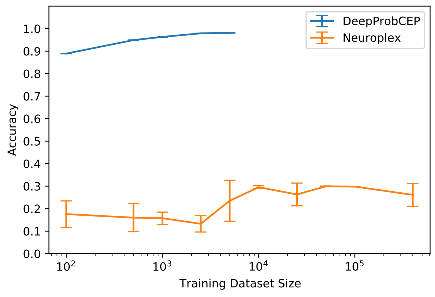
The errors shown by Neuroplex, the LSTM and the C3D are all related to the null class, which indicates that correctly identifying whether a case is part of this class or not is the hardest part of the problem. This fact is also quite relevant in terms of the performance of the whole system as, both in real scenarios and in our testing dataset, the null class is the most prominent, since complex events tend to be rare. This is especially true for smaller window sizes, as considering less complex events makes it less likely that a specific pattern will occur. This means that performing well at classifying the null class is more important for the overall accuracy than correctly classifying every single type of complex event. The higher percentage of null classes in the validation dataset for smaller window sizes, combined with the difficulty of classifying the null class, might be rewarding the systems for defaulting to the null class when they are unsure of which class they should decide on, which would explain why most of the cases that are incorrectly classified select the null class. However, it is important to note that this is only an effect of selecting the best performing weights on the validation dataset, as the training dataset is balanced, with the null class appearing the same number of times as each of the other classes.

4.7.2 Performance on simple events

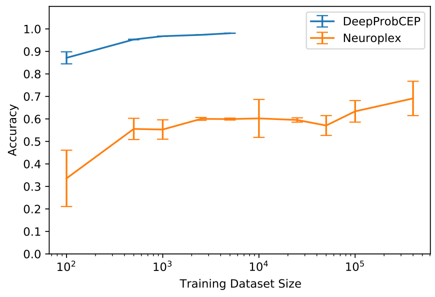
As discussed in Section 4.2.2, the neuro-symbolic approaches provide us with a simple event classifier as a byproduct of their training. This classifier can be obtained by extracting the low-level perception neural network from either of the neuro-symbolic approaches after training.

This is not possible in neural-only approaches, as they do not have an architecture separated into two layers. Furthermore, even if a similar architecture using a perception layer was used, the lack of human knowledge would prevent it from being able to learn to classify the simple events in a human intelligible way, as shown in [Xing et al., 2020]. Instead, the neural network would simply create its own representation of the simple events and rules, which would not be human-interpretable.

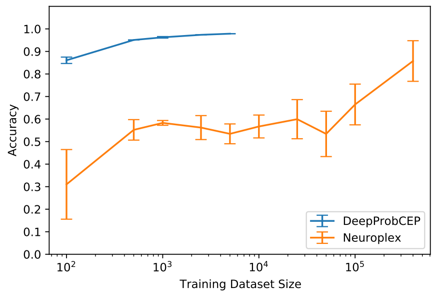
In this section, we evaluate the accuracy of the low-level perception neural networks from the neuro-symbolic approaches when classifying MNIST digits. Of course, this would not be possible in a real-world scenario if we are assuming that we have no labelled data for the simple events at all. However, this will give us an idea of how accurate we can expect these neural networks to be, compared to the accuracy of the system as a whole.



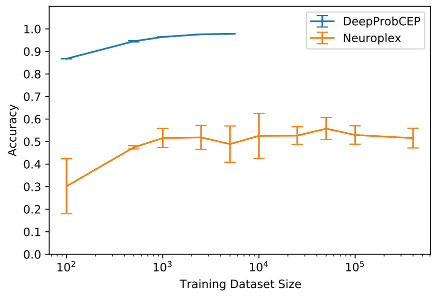
(a) Window of 2



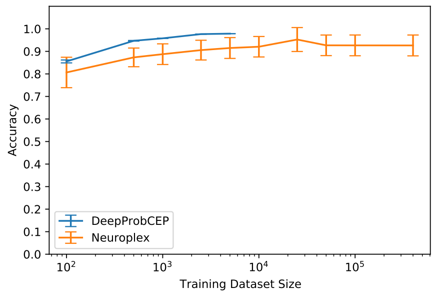
(b) Window of 3



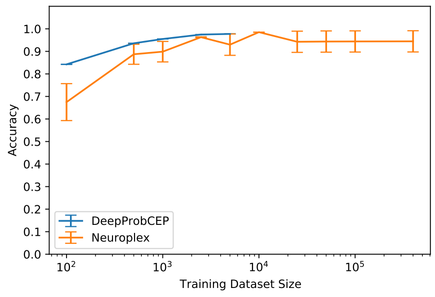
(c) Window of 4



(d) Window of 5



(e) Window of 10



(f) Window of 15

Figure 4.15: Accuracy for individual digits for each of the considered approaches in different windows. Horizontal axis indicates the size of the training dataset used. Graphs show average results over 3 different executions, with the standard deviations shown as error bars.

In Figure 4.15, we can see a comparison between the performances of DeepProbCEP and Neuroplex on single events. Again, the performance with Neuroplex does not seem to be very consistent with smaller windows. This would seem to furhter reinforce the hypothesis that Neuroplex does not work well with smaller window sizes. By comparison, we can see that DeepProbCEP performs quite well in any window size, especially when trained with 500 training points or more.

Furthermore, Neuroplex seems to have a relatively high standard deviation when compared to DeepProbCEP, even in the windows of 10 and 15 where Neuroplex seems to perform the best.

This matches the results we saw for the system as a whole, which would seem to indicate that, at least for the problem we are looking at, DeepProbCEP provides more consistent results. This could be expected, as (assuming the rules are correctly defined) the performance of the entire DeepProbCEP system should always correlate to the performance of the perception neural network used. This is because correctly detecting each of the simple events that constitute a complex event should always mean that the complex event is detected correctly. On the other hand, misclassifying one of the simple events will likely mean that the complex event is not detected correctly. The performance of the system as a whole would only be different from the performance of the perception layer if the reasoning layer is not consistent with how the simple events are aggregated into complex events in reality. With this in mind, we can also consider DeepProbCEP more robust, as the inconsistent performance of Neuroplex on the perception layer could indicate that the system is not able to consistently emulate the logic layer when using smaller window sizes.

4.7.3 Time efficiency

However, as indicated above, one of the main disadvantages when using probabilistic logic programming in the architecture is a substantial decrease in speed when compared to a neural network. This means that DeepProbCEP is significantly slower in both training time and inference time. This is the reason why we have not tested how DeepProbCEP performs after training with the bigger dataset sizes, as this would take a significantly longer time than with other approaches.

The exact difference in time-efficiency between DeepProbCEP and other approaches can vary depending on a number of factors, with the most important one seeming to be the window size of the dataset. Table 4.6 shows, for different window sizes, how many training iterations can be performed by each of the neuro-symbolic approaches (meaning that higher numbers are better). As shown in Table 4.6, bigger window sizes tend to be slower, as there is more data to process. Furthermore, for the DeepProbCEP approach, the logic layer will also increase in complexity as the window size increases, which will also decrease the speed. The same is true for the high-level reasoning neural network used in Neuroplex, but the effect seems to be slightly lower, as the speed decreases at a slightly slower rate. The results show that

Neuroplex is roughly 15 to 25 times faster than DeepProbCEP on training time. This is because Neuroplex is using a neural network to perform the high-level reasoning, which is implemented using Tensorflow and Keras frameworks. This allows Neuroplex to make use of the highly optimized back-end code from these libraries, as well as easy access to batch training and GPU use to increase the speed of training. In contrast, DeepProbCEP makes use of DeepProbLog for high-level reasoning, which does not currently support batch training and is significantly less optimized for fast training than Tensorflow or Keras. As we have said, the most significant cost in terms of time comes from using probabilistic logic programming. More specifically, the cost of calculating the output probabilities is based on the input probabilities that come from the prediction made by the neural network. In order to correctly calculate these output probabilities, the system needs to transform the logic defined by the user into an arithmetic circuit. Due to the current design of DeepProbLog, this transformation needs to be performed every epoch for every training case. As this transformation into an arithmetic circuit is, by far, the most time-consuming step of the training process, this is a very significant time cost.

DeepProbLog does offer a cache system that allows us to generate this arithmetic circuit on the first epoch and then re-use it on further epochs. Our experiments have shown that using this cache system can increase the speed by as much as 4 times, and potentially more if combined with other improvements. While this would be slower than Neuroplex, the difference would be much less significant. However, this approach requires us to keep a cache with all of the arithmetic circuits in memory. This has prevented us from using it on the bigger datasets, such as data-scarcity datasets of a size bigger than 5,000. This is because the amount of memory needed exceeds the limits of the RAM in our server. Therefore, we believe that further research is necessary to make the most out of this approach to increase the speed of the system.

It is worth noting that this cache system is a similar approach to PreCompilation (described above in Section 3.7.2), where the arithmetic circuits are re-used to skip the compilation step, thus significantly reducing the amount of time needed to perform the logical inference. However, the main difference between the PreCompilation approach and this cache is the fact that PreCompilation allows the user to re-use the same arithmetic circuit for different timestamps, making it possible to generate a small number of arithmetic circuits, or even just a single one. In contrast, this is not supported by the cache system, meaning that we need to save an arithmetic circuit for each individual query. However, combining these approaches seems like a very promising area for future research, which could allow for much faster training times when using DeepProbCEP.

	Window Size					
	2	**3**	**4**	**5**	**10**	**15**
DeepProbCEP	33.62	23.01	17.31	13.22	6.63	4.39
Neuroplex	500.00	333.33	333.33	333.33	142.86	111.11

Table 4.6: Training time-efficiency in iterations per second. That is, number of training points that can be processed each second (higher is better). Values obtained by dividing the number of data points in the training dataset by the training time for a single epoch of training. The data-scarcity dataset of size 2500 was used for this experiment.

4.8 Reliability

In this section, we explore the *reliability* of DeepProbCEP in regards to the definition provided in Chapter 1. That is, the quality of being trustworthy and performing consistently well. However, this section will not go into evaluations of *robustness*, which are one of the main focuses of the following chapter.

As shown in the results above, DeepProbCEP is the most reliable of the systems compared, performing well after training under any window size and with any of the dataset sizes. Furthermore, even when Neuroplex gives a similar performance (Window sizes of 10 and 15) the standard deviation for DeepProbCEP's performance is significantly smaller, meaning that DeepProbCEP gives a more consistent accuracy across the 3 training executions shown in the results. As such, we feel confident saying that DeepProbCEP is a reliable approach to CEP when using non-symbolic input data.

Another advantage of using DeepProbCEP in terms of reliability is the fact that it directly uses a logic layer, instead of trying to emulate it (as Neuroplex does) or disregarding it entirely (as the neural-only approaches do). Assuming the logic rules are correctly defined by the user, directly using them removes any possibility of a miss-classification due to an error on the reasoning. As such, any incorrect classification can only be caused by the perception layer incorrectly detecting which simple event has happened at a certain point in time.

On the other hand, neural-only approaches do not use any user-defined rules, meaning that these systems need to learn under which conditions a complex event is generated. Besides the increase in training data required, as shown above, this also opens the possibility that the system will incorrectly learn when a complex event is generated. This could be particularly concerning if the conditions for such a complex event are very specific, as the system may miss part of the restrictions and detect the complex event under circumstances where it should not, or if they are very broad, as the system may not have training data for all the possible combinations that generate such a complex event.

This issue is not as significant in the case of Neuroplex, as it takes into account the rule definitions for the complex events. This means that it will generally know under which conditions a complex event needs to be generated. However, the fact that a neural network (the NRLogic model) is used to emulate the function of the logic layer does introduce the potential that this NRLogic model will behave unexpectedly. While this could likely be corrected by generating more synthetic training data for the neural network that emulates the logic layer, the only way in which it can be detected is by testing every possible combination of inputs. This is infeasible when the number of combinations is high, making it effectively impossible to ensure that the neural network will perfectly emulate the logic layer. Testing every possibility is particularly difficult to do when using bigger window sizes. For instance, under the conditions shown above, testing how the logic layer behaves under each possible combination when using a window of 2 is effectively immediate, as only 10^2 possibilities need to be evaluated. On the other hand, performing the same test on the window of 5 required almost a minute, as 10^5 possibilities need to be evaluated. Finally, performing the test for bigger window sizes becomes infeasible, with expected total times of months for a window of 10 (over 1800 hours) and tens of thousands of years for a window of 15 (over $2 * 10^8$ hours). Of course, this is also a problem when training the NRLogic model, as it is infeasible to train it with all the possible combinations when using bigger window sizes. For our experiments in this chapter, we have used the default value of 10^5 synthetically generated cases to train the NRLogic model, which is significantly smaller than the 10^{10} and 10^{15} possible cases that it may encounter when using window sizes of 10 and 15. As such, it is very likely that the NRLogic model will encounter situations in testing that it has not encountered before. While we have not detected the NRLogic model predicting a logically incorrect result, there is the potential that a specific combination of inputs could cause that. This seems even more likely if more complicated and specific rule definitions are used. As such, it is harder to ensure that Neuroplex will perform in a reliable manner under changing circumstances.

4.9 Conclusion

As shown in the sections above, DeepProbCEP is the most consistent system when using sparse training data, providing the best performance for all experiments that use small amounts of training data by a significant margin. DeepProbCEP is also able to provide good performances with training datasets of intermediate size, consistently performing better than any of the other approaches. As a result, we feel confident saying that we have archived our objective of creating a system able to be data-efficient.

On the other hand, particularly when using larger window sizes (10 and 15), Neuroplex has a comparable performance with a much faster training time (between 20 and 25 times faster). This means that, in situations where getting a model working quickly is important, Neuroplex may be the better option. Furthermore, the faster training times can allow Neuroplex to learn from bigger datasets in the same amount of time.

We have also demonstrated that neural-only approaches can learn to classify complex events when given enough data. However, they require much larger quantities of training data to give comparable results, meaning that they are not a good choice when data is sparse. As shown in the experiments from Section 4.7.1, even in the simpler cases of window sizes between 2 and 3, neural-only approaches require at least 100,000 training samples to archive an accuracy of above 90%, while DeepProbCEP performs similarly when training with only 100 of them. As the window size increases and the problem increases in complexity, neural-only approaches perform even worse in comparison, where they are sometimes unable to reach an accuracy of 80% even with 400,000 training samples. Meanwhile, neuro-symbolic approaches, and DeepProbCEP in particular, perform significantly better with a fraction of those training samples, consistently being more than 80% accurate for any dataset with 500 training samples or more.

Furthermore, neuro-symbolic approaches have the advantage of providing us with a neural network capable of classifying simple events into the classes that we have defined. While this may not be a priority in all cases, it does allow us to use these neural networks on other projects. This neural network can also be used to evaluate how well the system is working, as the end-to-end performance of a neuro-symbolic approach should correspond to the performance of the perception layer. This is because neuro-symbolic approaches assume that the reasoning layer has been correctly defined by the user (or trained to match the definitions given by the user in the case of Neuroplex). On the other hand, if the reasoning layer is performing well but the system overall is not, this will usually be due to a mistake in the complex event definitions by the user, which results in the reasoning layer applying rules that are not consistent with how complex events actually happen.

It is also worth noting that, while ProbCEP has not been considered for this chapter, as it does not train its neural networks, it is possible to combine the ProbCEP and DeepProbCEP approaches. This is thanks to the fact that both approaches use ProbLog to define the complex event rules, combined with neural networks that provide the simple events. Combining the ProbCEP and DeepProbCEP approaches can allow users to use pre-trained neural networks to identify part of the complex events, while training other neural networks for the parts where no pre-trained neural network is available. For instance, in [Apriceno et al., 2021], authors use a pre-trained object detector to identify the position of MNIST digits in an image, while DeepProbLog is used to train a neural network to identify the value of those MNIST digits. In this case, the use of a pre-trained object detector is required as DeepProbLog restrictions currently make it impossible to train such neural networks. However, by combining the approaches, the system is capable of learning to classify the MNIST digits and, as a result, to identify the complex events.

4.9.1 Limitations and Future Work

Despite the advantages provided by the neuro-symbolic approaches, they do have some disadvantages that can make them less appropriate for certain types of problems. For starters, there is a clear limitation in the fact that users need to provide correct rules defining when the complex events happen. If no rules are provided, these approaches have no advantage over the neural-only approaches. On the other hand, if the provided rules are incorrect, the system is likely to learn how to classify the complex events incorrectly, or not be able to learn at all. Of course, the level to which this could be a problem depends on how incorrect the rule definitions are. As such, access to experts that can define the rules specifying when complex events must be generated in an accurate manner is a strong requirement for this type of approach. This means that such approaches could not be used if the rules for the complex events are not known.

A possible area of future research which could reduce this requirement could be a system that is able to automatically learn the rules for the complex events. While we have not explored this area in depth—which is why this thesis does not go into details for this approach— we had some initial success combining FastLAS [Law et al., 2020, Law et al., 2021], an Inductive Logic Programming (ILP) approach, with DeepProbCEP for a demonstration. For this demonstration, a small dataset was used to learn the rules using FastLAS. These rules were then used to define the complex events in DeepProbCEP, which allowed us to train the neural network inside the system [Preece et al., 2021]. While further research would be needed to improve the pipeline for general use, an approach like this could allow users to train the system without requiring experts to define rules that exactly express when complex events occur.

Outside the area of CEP, some recent research has also started to look at combining neuro-symbolic approaches with ILP [Dai and Muggleton, 2021, Cunnington et al., 2022]. These approaches aim to train a neural network and learn the symbolic rules of a neuro-symbolic approach at the same time. This is done by dividing each epoch in two steps; one where the neural network is trained and one where the hypothesis (that is, the current symbolic rules) is updated. After enough training epochs, the system should learn the correct hypothesis and train the neural network to extract the relevant symbolic information. This is a very promising area of research for the type of problem we are considering in this thesis, but the novelty of this research area makes it difficult to determine the full capabilities of such approaches. As such, further research is needed to apply such approaches to solve CEP problems.

Another area in which DeepProbCEP is limited is in terms of training speed. As explained above, the need to perform the logical inference for each iteration significantly reduces the speed at which the system can train. However, further research into the use of approaches such as PreCompilation could reduce the impact of this limitation. Other approaches include improving the speed of performing the inference itself. For instance, in [Sakama et al., 2021] authors use mostly linear algebraic operations on tensors to compute a logic program. While more research is required in this area, this could allow for a more efficient computation and make it easier to perform calculations in parallel, making the most out of multi-threaded CPUs and GPUs. At the moment, however, the fact that non-linear operations are required to compute the logic program and perform the inference means that the process is not differentiable, meaning that it cannot be used to train a neural network in a neuro-symbolic approach.

As discussed in Section 2.4.1, some approaches do incorporate logic rules into the neural network [Diligenti et al., 2017, Donadello et al., 2017]. This is usually done by defining the loss function in a way that encourages the system to fulfill the logic rules, usually through some type of penalty for each rule that is not followed. This, however, is a soft constraint, meaning that, while the system is encouraged to fulfill as many of the rules as possible, it does not necessarily fulfill all of them. In practice, what this means is that some of the manually defined rules may not be entirely correctly represented in the final system. As such, these approaches are ideal if speed is the priority and fully following the defined rules is not a strong requirement. However, they are not as desirable if users want to know for sure that the system will perform exactly according to the rules they have defined.

Finally, another limitation is the fact that neuro-symbolic approaches rely on the performance of the perception layer, the performance of which cannot be tested directly. This is because we are assuming that we are in a situation where we have no labelled data for the simple events, having only labels for the complex events. Assuming that the reasoning layer is defined correctly, the performance of the system as a whole should depend directly on the performance of the perception layer. As such, if the system is performing correctly, it is reasonable to assume that the perception layer is as well. However, as shown by the performance of Neuroplex in smaller window sizes (between 2 and 5), this is not necessarily always the case. In the case of Neuroplex, this is likely due to an unexpected output by the neural network emulating the logic layer, but there could be other causes. A way to ensure that the perception layer is performing correctly would be to generate a small testing dataset for it, with labels for the simple events. While this would be an added cost, this would be significantly less costly than creating the bigger training dataset that would be required to train the perception layer individually.

Chapter 5

Evaluating DeepProbCEP's Robustness and Agility

In the previous chapter, we demonstrated the capabilities of the DeepProbCEP approach to train with sparse data, making it *data-efficient*, and its *reliability* in terms of consistent accuracy performances and a sound logic layer. In this chapter, we will further explore the reliability of the DeepProbCEP approach by evaluating the *robustness* of a system against different poisoning attacks. That is, the ability of the system to continue to operate correctly despite adversarial circumstances. We also evaluate the *agility* of the DeepProbCEP approach, or the ability to change quickly and easily to solve new problems.

In order to evaluate the robustness of the approach, we continue to use the system used in the previous chapter for detecting complex events from an MNIST input stream. This is because this situation allows us to have great control over the noise in the training data, thanks to the fact that we are using synthetic datasets. To test the system's robustness, we train it using a number of synthetic datasets, each of them under different types of attacks and known percentages of noise. This allows us to compare how the system performs after training under each of those situations, giving us an idea of how it may perform when similar situations are encountered in the real world.

After evaluating the robustness of the approach, we also want to demonstrate that DeepProbCEP is agile, allowing for quick and easy changes in its behaviour. For that purpose, we describe how different frameworks can be used for the definition of complex events, allowing users to easily specify exactly when they want the complex events they are interested in to be detected. We also describe the functionalities and internal design for the sequence framework, which was used to define the rules for the system that detects complex events from a stream of MNIST digits.

Another way in which DeepProbCEP is agile is by offering users the possibility of changing the behaviour of the system after training. We divide this change of behaviour into two sections: one focusing on changes to the reasoning layer, and one focusing on changes to the perception layer. Thanks to the use of DeepProbLog, changes to the reasoning layer are as simple as changing the rules for the complex events and loading the already trained weights for the perception layer. On the other hand, changes to the perception layer do require some further training. However, we explore how fine-tuning can be used to reduce the amount of time required to train the system, while providing similar performances as training the system from scratch.

We also show how the DeepProbCEP approach allows for the detection of complex events from different types of input data. To demonstrate this, we use a stream of audio as an input, which not only shows that DeepProbCEP can be used with more challenging types of input, but also brings us closer to types of data that could be useful in the real world.

5.1 Robustness

In this section, we will explore how robust DeepProbCEP is when training with different types of poisoned datasets. For this purpose, we have generated a number of synthetic datasets with different types of poisoning attacks. We implemented the attacks in a manner that allows us to evaluate up to which percentage of noise still allows the system to train correctly, which will inform us on how systems with the DeepProbCEP approach may perform when training with real-world datasets.

Table 5.1: Summary of the dataset types used in this chapter.

Dataset Name	**Targeted**	**Substitute**
Random Noise	All	Randomly chosen
Poisoning Single Label	One	One
Poisoning Multiple Labels	All	One for each targeted class
Diluting	One	Randomly chosen
Strict Diluting	One	Randomly chosen excluding targeted class

Table 5.1 shows a summary of the attacks we have evaluated. To evaluate the robustness of our system, we have generated datasets for each of the attacks. In each case, one of the attacks is

used to poison the training dataset, which is done by changing the ground truth labels for an incorrect one. As shown in the table, the types of attacks differ in terms of how many classes are targeted, as well as how the substitute is chosen. The targeted classes represent which ground truth labels can be affected by the attack. For our experiments, this can either be all classes or a single class. Meanwhile, the substitute classes represent the possible values that the incorrect label may take. This can either be a pre-selected value or chosen randomly each time. Unless specifically stated, there is a possibility that the substitute class is the same as the targeted class when choosing randomly choosing the noisy label.

5.1.1 Dataset Generation

In the following sections, we explain how we have created the dataset types described above, as well as what type of situation each dataset is trying to emulate. Each dataset is affected by a different type of adversarial attack. In general, these datasets are trying to emulate situations that may happen with datasets in the real world, either due to human mistakes when labelling or deliberate attacks. Using synthetic datasets we can artificially introduce a known percentage of incorrect labels to the training datasets. This estimates how well our approach might perform when using a real dataset, which might contain an unknown level of noise.

Our synthetic datasets have been generated based on the MNIST dataset. For each type of attack, window sizes between 2 and 5 (both included) have been generated. All datasets are balanced and set to a fixed size of 2500 training points, which allows us to fairly compare the different approaches and window sizes.

Random Noise

This type of attack tries to emulate the fact that real-world complex events can be difficult to identify, and thus may be labelled incorrectly due to a human error. This can lead to errors in the training dataset, which might affect the accuracy of the system after training.

We have created datasets with different amounts of noise by randomly changing different percentages of the training labels for another random label. The noisy part of the dataset will have randomly assigned training labels, instead of the ones that should be assigned according to the complex event rules.

In order to generate the random noise datasets, we use the same steps described to generate the base dataset, explained above in Section 4.5.1, Algorithm 9. However, when labelling a certain timestamp as a complex event, there is a probability of changing that label to a random complex event. This probability is determined by the noise percentage parameter. We have generated datasets ranging from 0.0 to 1.0, with a step of 0.2. For this, 0.0 means that no data has been poisoned, while 1.0 means all the data has been poisoned.

This kind of noise might appear both due to a malicious agent, and to the difficulty of labelling the sophisticated scenarios where a system like DeepProbCEP would be useful: for instance, different annotators might be having different considerations on what constitutes the complex event.

Poisoning Single Label

This attack aims at emulating situations in which a complex event is confused with another type of complex event. This may be due to similarities between those types of complex events, or due to a malicious attack aiming to make the system miss-classify a certain complex event as another class.

Before starting the generation of the dataset, a complex event class is selected as the target. Then, a different complex event class is chosen as the substitute. During the generation of the dataset, any time the complex event rules indicate that the label should be the selected target, there is a probability of changing that label to the substitute class. The process for this is similar to the one explained above for the random noise datasets. However, for this case, there is a check that only allows for the label to be modified if, according to the ground truth, it should be labelled as the targeted class. As such, all training cases for other classes remain unchanged. As was the case for the random noise datasets, the probability of changing the label is determined by the noise percentage parameter, and we have generated datasets ranging from 0.0 to 1.0, with a step of 0.2. In this case, this probability only affects the targeted class. A noise of 1.0 would mean that all classes belonging to the targeted class are now incorrectly labelled as the substitute.

Poisoning Multiple Labels

For this attack, all complex events are targeted. In this case, we choose a substitute for each complex event, which must be different from the corresponding targeted complex event. This can be seen as randomly shuffling the labels for the complex events, ensuring that the shuffled labels do not land in their starting position. Table 5.2 shows an example of a possible substitution for the labels.

Targeted Label	ce0	ce1	ce2	ce3	ce4	ce5	ce6	ce7	ce8	ce9
Substitute Label	ce9	ce0	ce8	ce1	ce6	ce4	ce7	ce2	ce5	ce3

Table 5.2: Example of a possible substitution for Poisoning Multiple Labels.

This attack should cause the system to incorrectly classify all complex events as another type of complex event, based on the substitution we are applying. The process for generating the datasets under this type of attack is very similar to the previous attack. However, in this case, all complex event classes are targeted, instead of just one. Again, we use probabilities between 0.0 and 1.0. In this case, a probability of 1.0 would mean that all the training points have the substitute class label, effectively meaning that we are training for the incorrect class.

Diluting

This attack functions in a similar manner to the poisoning single label attack. However, instead of always substituting the targeted class for a previously decided class, the substitute is chosen randomly each time. Similarly to the poisoning single label attack, we would expect this to also lower the accuracy and confidence of the targeted class. We are interested in what kind of differences there will be between the two attack types.

In the real world, this type of attack could happen if the person labelling the dataset is unsure about how to classify one type of complex event. However, most cases will likely be due to a malicious attack intending to lower the performance for a specific class in a manner that is harder to notice than the poisoning single label attack.

Strict Diluting

This attack functions in the same way as diluting with the only difference that the randomly chosen substitute class cannot be the targeted class.

5.1.2 Experiment Design

In this section, we present the experiments performed using the datasets described above. In order to evaluate how robust our approach is against the types of attack from each of the datasets, we have trained a system using each of the datasets. Then, we compare the accuracy performance obtained across each of the percentages of noise.

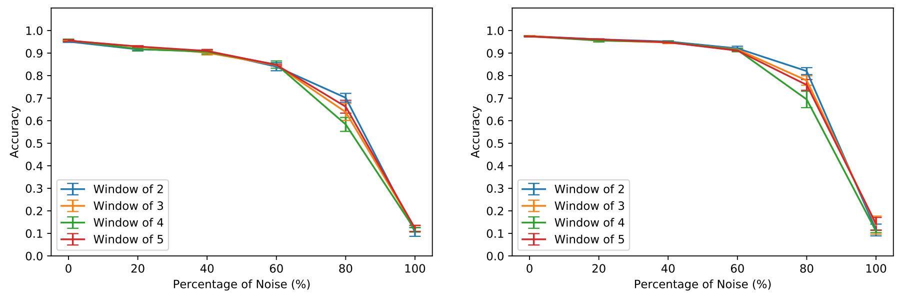
We consider a system to be *highly* robust against a type of attack as long as the performance does not decrease significantly for percentages of noise lower than 50%. As it seems unlikely that more than half of the training data will be labelled incorrectly without anyone noticing, highly robust approaches should be expected to perform well even if that type of attack is present. We consider the system to be *sufficiently* robust against that type of attack as long as the performance does not decrease significantly when using a training dataset where 20% of the data is noisy. We believe that having a dataset where 20% of the training data is noisy is conceivable, while still being a high enough percentage that the performance could be affected.

It is worth noting that the systems are evaluated using the testing datasets, which are unaffected by the attack. This means that the ground truth labels in the testing dataset are correct, to allow us to evaluate how much each of the attacks would affect the performance of the system in a real-world scenario.

5.1.3 Results

In the following sections, we will evaluate how robust DeepProbCEP is against the datasets described above. For all the graphs, the system has been tested with levels of noise between 0.0 and 1.0 (both included) with a step of 0.2. However, in order to make it easier to see the error bars in the graph, these have been slightly offset in the x axis from their actual value.

Random Noise



(a) Complex Event Accuracy (b) Digit Accuracy

Figure 5.1: Accuracy results for Random noise.

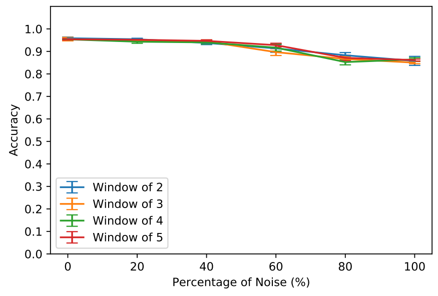
In Figure 5.1 we explore how the performance of DeepProbCEP is affected by random noise in the training data. As can be seen in the graphs, DeepProbCEP performs very well even with high percentages of noise. The accuracy does start to decrease when the percentage of noise is above 50%, but, surprisingly, still remains acceptable until 80% of the data is noisy. As such, we consider the system highly robust against this type of attack.

The graphs also show a correlation between the accuracy when detecting the complex events and the accuracy on the simple events. This could have been expected, as DeepProbCEP fully relies on the performance on the simple events to correctly predict the complex events, assuming that the rules are correct.

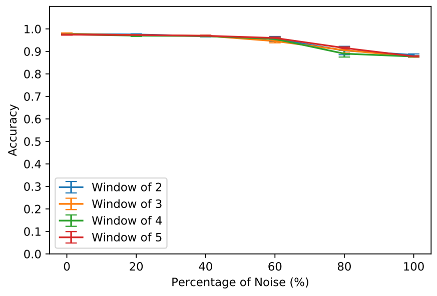
Finally, we can see that the performance when training with 100% of noise is very close to zero in all cases. This could have been expected, as datasets with 100% of noise effectively give no useful training data, which leads to DeepProbCEP behaving similar to a random classifier.

Poisoning Single Label

Next, we consider the targeted attack for a single label, where only one of the classes is affected by the attack. As we can see in Figure 5.2, the performance for both the system as a whole and the individual digits is almost unaffected. In fact, even the performance on the dataset with a noise percentage of 1.0 might seem acceptable for some situations. This is because only one of the classes is being affected, meaning that the other 10 classes (the 9 other patterns plus the null class) are still performing the same. However, one of the classes is being incorrectly classified



(a) Complex Event Accuracy

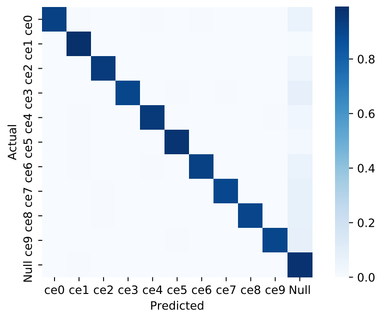


(b) Digit Accuracy

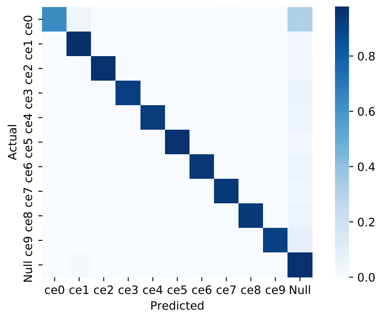
Figure 5.2: Accuracy results for Poisoning Single Label.

a fairly large percentage of the time, which would significantly affect how the system performs in a real-life scenario. This can be more clearly seen in the confusion matrices in Figure 5.3, which show the results of this attack when using a window of 4. The confusion matrices show that the targeted class (in this case, $ce0$) tends to be miss-classified, either as the substitute class ($ce1$ for this scenario) or the null class (shown in the last column and row). When training with higher noise percentages (Figures 5.3c and 5.3d), the targeted class tends to be classified as the substitute class most, if not all, of the time. This is because, with high levels of noise, we are basically training the system to classify the targeted class as the substitute. Meanwhile, for lower percentages of noise (Figures 5.3a and 5.3b) the targeted class is either classified correctly or miss-classified as the null class. This is because the training is affecting how the perception layer classifies the simple events. For lower noise percentages, the perception layer will still classify the simple events correctly a significant percentage of the time. As such, in these situations, it is likely that at least one of the simple events will be classified correctly. This means that, when the rules are evaluated, the system will either identify both simple events correctly, thus identifying the complex event correctly, or miss-classify only one of the two simple events that generate the complex event, meaning that no complex event happens according to the rules. Of course, it is also possible that both simple events will be miss-classified, but, as long as the percentage of noise is low, this is not as likely.

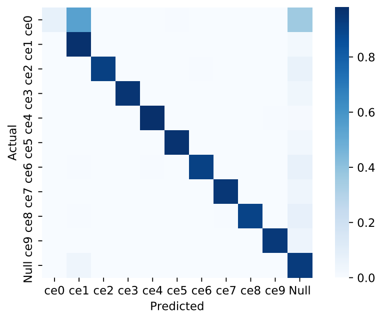
This type of attack shows that it is possible for our system to perform badly in specific cases without significantly affecting the overall performance. This means that a good overall performance does not necessarily ensure that all of the classes are performing well. To detect such cases, we can look at the confusion matrices or at the recall for each of the classes, which



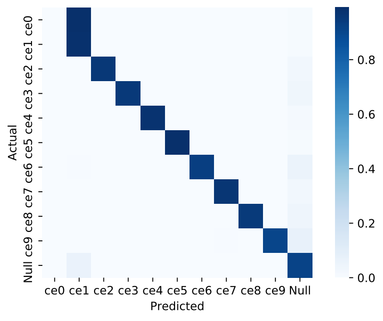
(a) 0.4 Noise



(b) 0.6 Noise



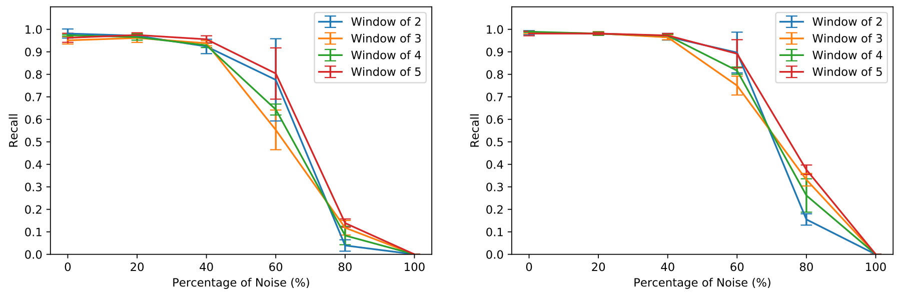
(c) 0.8 Noise



(d) 1.0 Noise

Figure 5.3: Confusion matrices after training under the Poisoning Single Label attack. All confusion matrices show results for a window of 4. For this case, the targeted class was $ce0$ and the substitute was $ce1$.

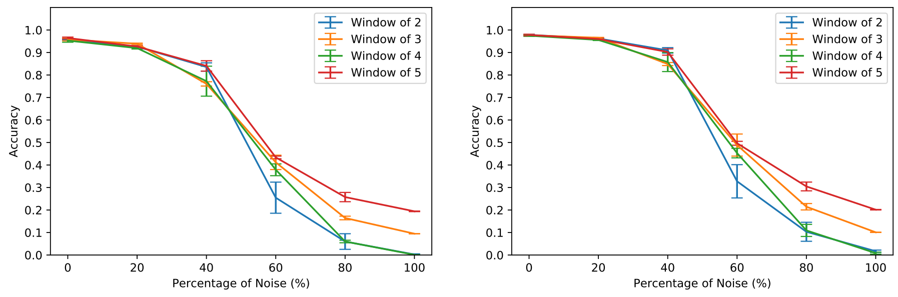
provides us with a value that represents the performance for that class. Figure 5.4 shows the recall values for the targeted class as the percentage of noise increases. Figure 5.4 shows that increasing the percentage of noise quickly reduces the recall performance for the targeted class. This means that, if we want to make sure that all classes perform correctly, it will be necessary to evaluate them individually. Despite this, both the accuracy and recall remain quite high for percentages of noise up to 40%, meaning that we still consider the system sufficiently robust against this type of noise.



(a) Complex event recall (b) Digit recall

Figure 5.4: Recall results for the targeted simple and complex events under the Poisoning Single Label attack.

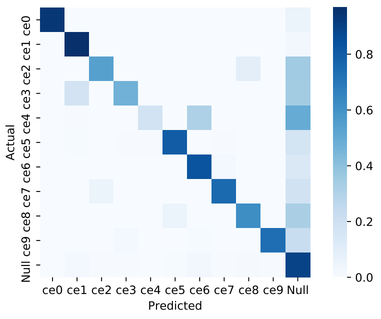
Poisoning Multiple Labels



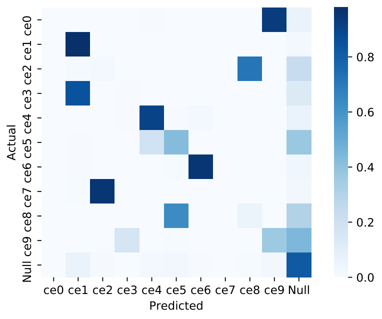
(a) Complex Event Accuracy (b) Digit Accuracy

Figure 5.5: Accuracy results for Poisoning Multiple Labels.

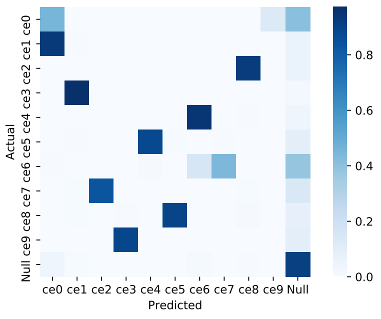
Now we will look at the poisoning multiple labels attack, the results of which can be seen in Figure 5.5. As can be seen in the graphs, the performance for DeepProbCEP rapidly decays as we increase the percentage of noise present in the training data. This could have been expected since, again, what we are doing once the noise percentage is higher than 50% is training the neural network to recognize the incorrect label. This can be seen quite clearly in the confusion matrices shown in Figure 5.6. Particularly in Figure 5.6d, where all classes are consistently being classified as their substitute.



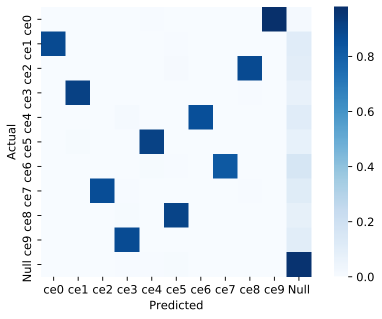
(a) 0.4 Noise



(b) 0.6 Noise



(c) 0.8 Noise



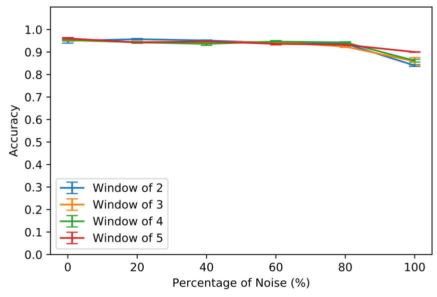
(d) 1.0 Noise

Figure 5.6: Confusion matrices after training under the Poisoning Multiple Labels attack. All confusion matrices show results for a window of 4.

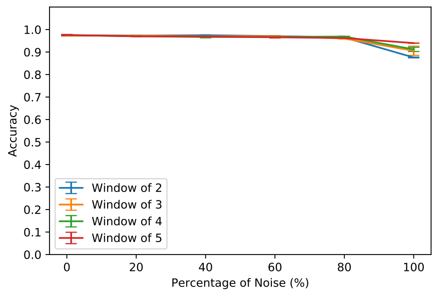
However, it seems unlikely that such a big percentage of incorrect labels would make it to the training dataset without anyone realizing it, as most of the training data would need to be consistently incorrectly labelled for this to happen. In contrast, for reasonable amounts of noise levels (such as 20% and, to a certain degree, 40%) the system still performs fairly well, which shows us the system is sufficiently robust against this type of attack.

Diluting

Figure 5.7 shows the results for the Diluting attack on complex event accuracy. Similarly to the Posioning Single Label attack, the overall accuracy remains almost unchanged despite high levels of noise. However, if we look at the recall for the targeted class, shown in Figure 5.8, we can see that high levels of noise do significantly reduce the performance of that class. In this case, however, the performance starts to decrease with a higher percentage of noise, particularly

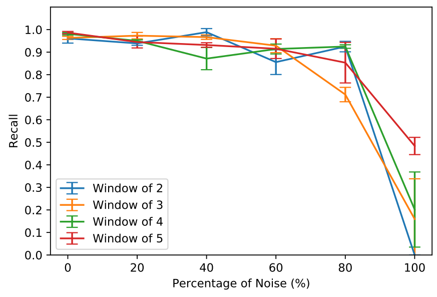


(a) Complex Event Accuracy

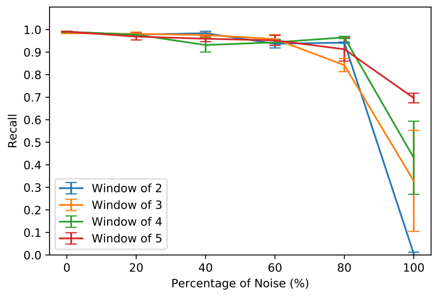


(b) Digit Accuracy

Figure 5.7: Accuracy results for Diluting.



(a) Complex event recall

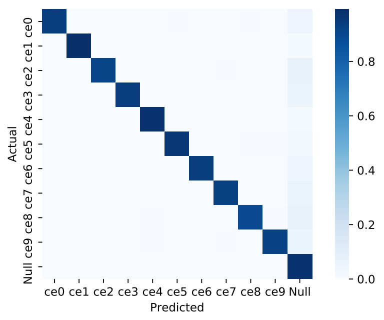


(b) Digit recall

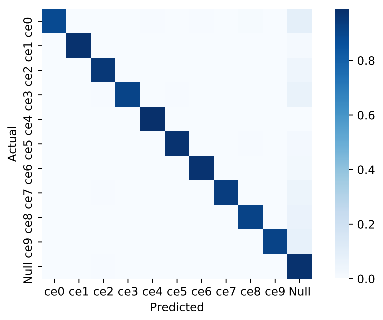
Figure 5.8: Recall results for the targeted simple and complex events under the Diluting attack.

in the MNIST setting. This is reflected in the confusion matrices, shown in Figure 5.9, where the targeted complex event ($ce0$) is mostly classified correctly in all cases except with 1.0 noise.

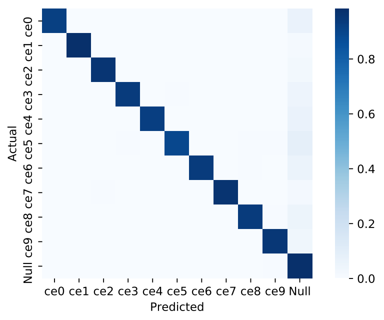
This is likely because this type of attack is less focused on miss-classifying the targeted class as one specific other class. Instead, this attack tends to reduce the confidence of the system for the targeted class. This does mean that a higher percentage of the data needs to be noisy for the attack to have an effect. However, it may also make the attack harder to detect than consistently substituting the value for one label. Despite this, we would still consider the system to be highly robust against this type of noise, as the system still maintains both the overall accuracy and the recall for the targeted class well over 50% of noise.



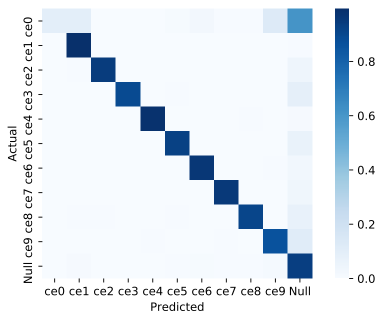
(a) 0.4 Noise



(b) 0.6 Noise



(c) 0.8 Noise



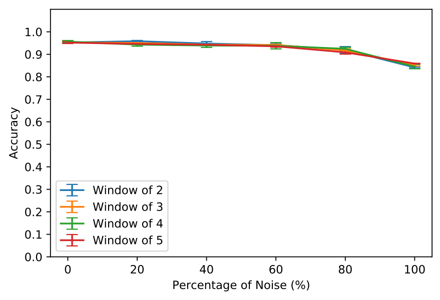
(d) 1.0 Noise

Figure 5.9: Confusion matrices after training under the Diluting attack. All confusion matrices show results for a window of 4. For this case, the targeted class was $ce0$.

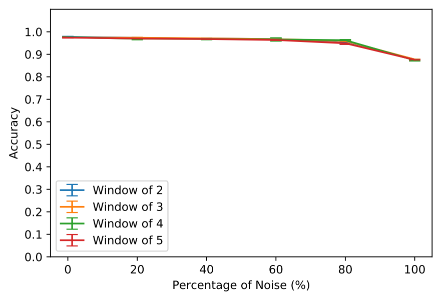
Strict Diluting

Figure 5.10 shows the accuracy results for the strict diluting attack. As could have been expected, they are quite similar to the results for the diluting attack due to the similarities between these two attacks.

However, looking at the recall results in Figure 5.11 and the confusion matrices in Figure 5.12 we can see that the performance for the targeted complex event is slightly more affected in lower levels of noise. This is due to the fact that none of the noisy data has any chance of randomly getting the correct label, unlike the previous attack. However, even in this case, we would consider the system to be highly robust to this type of attack, as almost all of the training data needs to be noisy to affect the performance of the system.

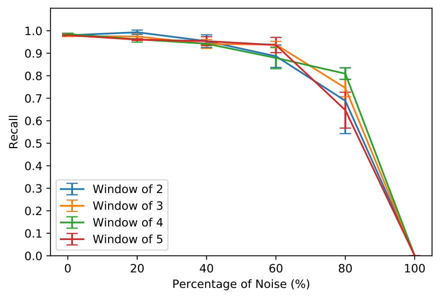


(a) Complex Event Accuracy

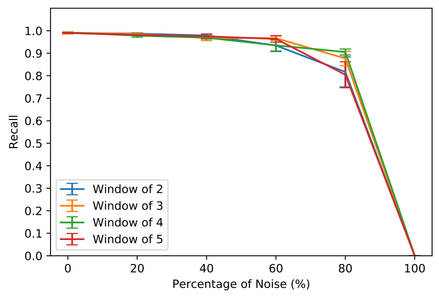


(b) Digit Accuracy

Figure 5.10: Accuracy results for Strict Diluting.



(a) Complex event recall

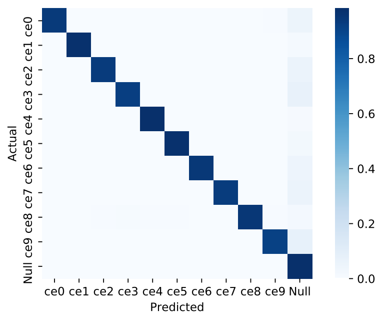


(b) Digit recall

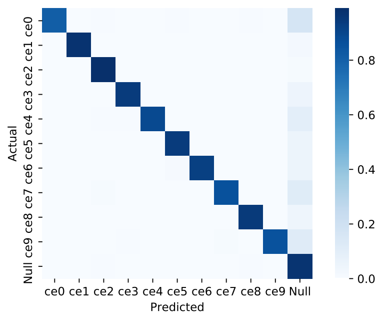
Figure 5.11: Recall results for the targeted simple and complex events under the Strict Diluting attack.

5.1.4 Robustness Evaluation

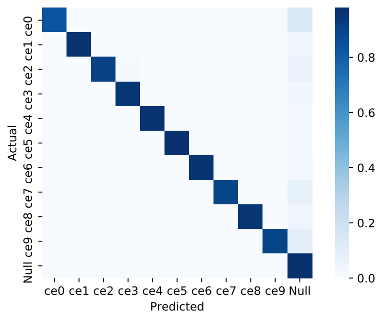
DeepProbCEP is highly robust against most types of attacks, with performances remaining high even when more than 50% of the training data for the targeted classes is noisy. As such, we feel confident that similar systems should perform well when dealing with the noise that may be found in real-world datasets, as it seems unlikely that higher percentages of noise may appear in a training dataset without being noticed.



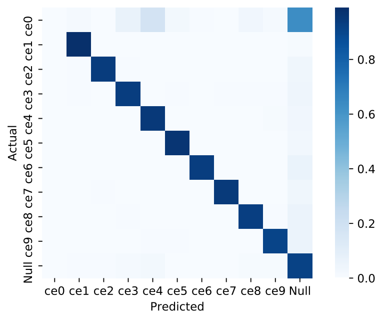
(a) 0.4 Noise



(b) 0.6 Noise



(c) 0.8 Noise



(d) 1.0 Noise

Figure 5.12: Confusion matrices after training under the Strict Diluting attack. All confusion matrices show results for a window of 4. For this case, the targeted class was $ce0$.

However, we have also shown that, in some cases, it is not enough to evaluate the overall accuracy to detect certain types of noise. As explained above, this is due to a combination of the facts that there are many types of complex events and that the number of instances where no complex event occurs in the testing dataset are much higher than the other classes. While this is done to reflect the situation in the real world, it can give a biased perception of the performance of the system. As shown above, evaluating the recall for each of the classes can be a useful tool to detect these situations.

5.2 Agility

In the previous sections, we demonstrated the robustness of a system that uses the DeepProbCEP approach. For the rest of the chapter, we will focus on exploring how systems that use the DeepProbCEP approach can be agile.

As defined in Chapter 1, an agile approach is one that allows for quick and easy changes. In the following sections, we explore different properties of the DeepProbCEP approach that increase its agility, making it possible to adapt its behaviour to many different situations in a simple manner.

First, we explore how DeepProbCEP allows for the use of frameworks, which allow the user to easily define when the complex events are happening in an easy manner.

Then, we discuss how systems created using the DeepProbCEP approach can be refined if further information about the complex events is obtained after training. This refinement can result in an improvement in the accuracy when detecting such complex events. For this purpose, we consider two types of changes. Initially, we explore how the reasoning layer can be modified after training, and changed to specify the conditions under which complex events are generated more accurately, or even detect entirely new complex event types.

Then, we explore how the perception layer can be fine-tuned after training to add simple event types that were not considered in the initial training. This can allow us to use simple events that more accurately represent the input data when defining the conditions under which a complex event is generated, which should result in more accurate detection of the complex events.

5.2.1 Frameworks

One way in which the agility of DeepProbCEP is increased is through the use of frameworks, which allow the user to define the rules that specify when complex events happen in different ways. For instance, users could use the *event calculus framework* defined above in Section 3.3.3, which allows the user to control when complex events start and end by defining a set of conditions in the form of rules. The framework then allows the user to perform queries on whether any given complex event is occurring or not at any given point in time, as it internally keeps track of the state of all complex events.

In the following sections, we explain the *sequence framework*, which allows users to define when complex events are happening by giving a pattern of simple events, which must happen within a given window size. Of course, further frameworks could also be defined to match the requirements given by the user.

It is also possible to combine multiple frameworks, allowing for more complex definitions. For example, in Section 6.2 below we demonstrate how the sequence framework can be combined with the event calculus-inspired complex event definitions.

In the following sections, we give a description of the different functionalities provided by the sequence framework, as well as how they have been implemented.

Functionalities Offered by Sequence Framework

This section explores the functionalities offered by the sequence framework. Three clauses are used for this purpose: `sequence`, `warps` and `exclude`.

`sequence` The main clause offered to users by the framework is `sequence(L, W, T)`, which allows users to define a list `L` of simple events. The clause will be true if this list `L` happens, in the provided order, within a given window `W` of time, also provided by the user. Finally, the value of `T` is the timestamp at which the sequence is evaluated. This is usually left as a variable in rule definitions, which allows the user to specify which timestamp they want to evaluate at run time. Both `W` and `T` are expected to be integers, whereas `L` should be a list.

Example 1. We are assuming that we have an input stream that periodically provides us with events indicating the temperature of the machine. We consider simple events that cover different temperature ranges (*lowTemp*, *mediumTemp*, *highTemp*, *extremeTemp*, etc.). For the reasoning layer, we are not concerned about whether these classes are the result of the classification performed by the low-level perception, or if the data from the input stream was already symbolic. This is because the logic will be the same whether the values come directly from the input stream or if they have been classified by the low-level neural network.

```
holdsAt(overheating, T) :-
    sequence([highTemp, highTemp], 5, T).

holdsAt(overheating, T) :-
    sequence([extremeTemp, extremeTemp], 5, T).
```

Listing 5.1: Example of a complex event definition.

Listing 5.1 shows a way in which the rules for *overheating* could be defined. First, we define the complex event *overheating* (line 1) to be formed by two instances of the simple event *highTemp* (line 2). This means that, if we have two high temperature simple events within the defined window of time, we will create an *overheating* complex event. For this case, the window of time has been set to 5. However, this could be changed to whatever is appropriate for the complex event. Lines 4 and 5 define that the complex event *overheating* can also occur if there are two instances of an extreme temperature within a window of 5. Using those rule definitions would allow us to perform queries of the form *holdsAt(overheating, T)*, where *T* can be substituted for any timestamp. For instance, performing the query *holdsAt(overheating, 10)* would return a value between 0 and 1. This value would represent the likelihood of the complex event *overheating* happening at timestamp 10. For instance, if we are confident that an instance of *extremeTemp* event has occurred at timestamp 10 and another instance has happened between timestamps 6 and 9, an *overheating* complex event would be generated based on the second rule.

`wraps` The sequence framework also provides a `wraps(A, B)` functionality. This allows the user to define a simple ontology where a single keyword can be used to refer to multiple simple event types. By defining the clauses `wraps(A, B)` and `wraps(A, C)`, the user is indicating to the system that `A` is a keyword that can be used to refer to both `B` and `C`. Furthermore, if the clause `wraps(G, A)` is added, the keyword `G` can be used to refer to `A`, `B` and `C`.

Example 2. In Example 1, we generate an *overheating* complex event if two instances of *highTemp* or two instances of *extremeTemp* occur. This, however, does not include cases in which an instance of each of them occurs. Given the problem we are trying to define, it would instinctively seem like that should be the case. One option to solve this would be to add two more rules, to take into account the two more possible combinations to generate the overheating complex event (*highTemp* followed by *extremeTemp*, and vice versa). However, this solution could quickly create a very high number of rules that are all trying to define the same situation. This would make the code needlessly long and complicated to maintain. This can be avoided by using `wraps`, as shown in Listing 5.2.

```
1 wraps(dangerousTemp, highTemp).
2 wraps(dangerousTemp, extremeTemp).
3
4 holdsAt(overheating, T) :-
5     sequence([dangerousTemp, dangerousTemp], 5, T).
```

Listing 5.2: Example of a complex event definition using `wraps`

Here, the user defines that both *highTemp* and *extremeTemp* are dangerous (lines 1 and 2). Then, using *dangerousTemp*, a single rule can be defined to represent all 4 possible combinations (lines 4 and 5).

`exclude` Another functionality offered by the framework is the possibility of excluding certain simple events from happening at specific positions in the window. By default, `sequence` allows for other simple events to happen between the simple events used in the sequence. This means that, in Examples 1 and 2, if we got the simple events *highTemp*, *lowTemp*, *highTemp* from the input stream, in that order, the complex event *overheating* would still occur. Of course, this is assuming that both instances of *highTemp* are within the given window of time. If we wanted to prevent the complex event from being generated in that case, the sequence framework allows the user to define that specific simple events are not allowed to happen between the other simple events. This can be done using the `exclude(C)` clause within the list `L` provided to `sequence`, which indicates that `C` cannot happen between the instances of the simple event defined before and after it in list `L`. If `exclude(C)` is used at the start of list `L`, then `C` cannot happen before the first simple event defined in the list, within the elements considered in the given window `W`. If it is used at the end of list `L`, it cannot happen between the last defined simple event and the evaluated timestamp. It is worth noting that the `C` defined in `exclude(C)` can also be a keyword defined using `wraps`, in which case all of the wrapped elements are excluded.

These rules also apply if multiple `exclude` clauses are defined in a row in list `L`. For instance, if `exclude(C1)` and `exclude(C2)` —assuming $\texttt{C1} \neq \texttt{C2}$— are used in a row neither `C1` or `C2` is allowed to happen for the period of time defined according to the rules above, as shown in the following example.

Example 3. Returning to Example 2, let us assume that we want to prevent the complex event from occurring if an instance of *freezingTemp* or *lowTemp* happens between the two instances of *dangerousTemp*. Listing 5.3 defines such a situation between lines 5 and 10.

```
1  wraps(dangerousTemp, highTemp).
2  wraps(dangerousTemp, extremeTemp).
3
4  holdsAt(overheating, T) :-
5      sequence([
6          dangerousTemp,
7          exclude(freezingTemp),
8          exclude(lowTemp),
9          dangerousTemp
10     ], 5, T).
```

Listing 5.3: Example of a complex event definition combining the use of `wraps` and `exclude`.

However, the definition from Listing 5.3 still allows simple events other than *freezingTemp* and *lowTemp* to happen between the two instances of *dangerousTemp*, unless they are specifically forbidden. For instance, unless we added `exclude(mediumTemp)` to the list between the instances of *dangerousTemp*, the system would still generate an *overheating* complex event if *mediumTemp* instances appear between the two instances of *highTemp*. This gives the user full control of exactly which simple events are allowed to happen within the sequence.

Internal design of the Sequence Framework

The full code defining the functionalities for this framework can be found in Appendix B. In this section, we provide a general description of how this code has been designed to implement the functionalities described above.

The main interface for the framework is the `sequence(L, W, T)` clause. Internally, this is defined using `sequenceEndingAt(LReversed, W, T)`, where `LReversed` is the reverse of `L`. This is done to simplify the rule definitions, as the code tries to start at timestamp `T` and work backwards in time to find all the simple events.

The clause `sequenceEndingAt` is responsible for matching the simple events we are expecting to happen according to the list with the simple events that happen in the given sequence. As such, it considers two possibilities:

- The first element is not using `excluding`. In this case, it checks whether that simple event happens at timestamp `T`. If it does, then the first simple event from list `LReversed` is removed and the clause `sequenceWithin` is used to detect whether the rest of simple events happen for the given timestamp and window. If the simple event does not happen at timestamp `T`, the sequence does not happen for that timestamp.
- The first element is using `excluding`. If so, `sequenceWithin` is used to deal with excluded simple events.

The clause `sequenceWithin` is responsible for evaluating whether the excluded simple events happen between the events we are expecting to happen. This is done by keeping a list of all excluded events. If an `exclude` is found in `LReversed`, the corresponding simple event is added to the list of excluded events. `sequenceWithin` also defines the base case for the recursivity. The base case will happen when all expected simple events form `LReversed` have happened in the right order, and, on top of that, at least one of the following is true:

- The list of excluded events is empty. This case is not strictly necessary but it makes cases where the `exclude` clause is not used as the first element of `L` faster to evaluate.
- We have looked at the entire window.
- We have reached the start of the timeline (timestamp 0).

If all expected simple events have happened, but none of the above is true, a recursive call will check each timestamp until the system has looked at the entire window or reached timestamp 0. For each of those timestamps, it will make sure that none of the simple events that happen is in the list of excluded events. Once one of the base cases is reached, the clause `sequence` will be true. If a simple event happens while being in the list of excluded events, the clause will be false through negation as failure.

If we have not found all the simple events defined by the user, we are still not in the base case for the recursivity. This will be the case as long as `LReversed` is still not empty. In that case, the clause `sequenceWithin` will consider three possibilities:

- The first element of `LReversed` is using the clause `exclude`. In that case, the element is removed from `LReversed` and added to the list of excluded events. After this, `sequenceWithin` is called recursively.
- The first element of `LReversed` matches the simple event happening at the timestamp we are currently looking at. If so, `sequenceEndingAt` is called, removing the element from `LReversed` and continuing the recursive call. This also clears the list of excluded events.
- Neither of the above apply. If so, the code checks that the simple event happening at the timestamp we are currently looking at is not in the list of excluded. If so, the recursive call continues. Otherwise, the clause is false through negation by failure.

Finally, to implement the `wraps` functionality, whenever a simple event that is wrapped by another happens, the wrapping event happens as well. This allows the code described above to use events without needing to consider whether they come from the input stream or from the wrapping functionality.

5.2.2 Changing the Reasoning Layer

Another way in which systems that use the DeepProbCEP approach can offer agility is by allowing changes after training. This can happen if further information about the complex events is discovered after deploying the system. This may require users to make changes to the system, to allow it to detect the complex events correctly.

These changes can be due to users gaining a better understanding of the complex event definitions, requiring an update to the rules to reflect this, or if users wanted to use a trained perception layer to detect other types of complex events, with entirely different rules. However, changes to the reasoning layer are restricted by the fact that the simple events used must remain the same. We consider the situation in which the perception layer needs to be changed in Section 5.2.3.

Experiment Design

The reasoning layer can be modified as much as the users want, without affecting the performance of the perception layer. The only exception is that the definition of the neural AD that forms the connection between the layer must remain the same. This means that the input and output of the neural network must remain the same from the point of view of the rules. Of course, this is because the functionality of the neural network must remain the same, as the perception layer will be re-used as it was, keeping the same trained weights.

Besides this constraint, the rules in the logic layer can be modified as much as the user wants. This is because the reasoning layer can consult the predictions from the perception layer as needed, but the two layers are otherwise independent. In order to demonstrate this, we have trained DeepProbCEP with one of the data-scarcity datasets, using 5,000 training samples and a window of 5. This dataset uses the same rules explained in the previous chapter, where $ce0$ happens when a 0 appears in the last position of the window and at least another 0 in any other position within the window, and so on for each complex event up to $ce9$. After training with this dataset, the system achieves a 97.5% accuracy when detecting complex events and 97.8% accuracy on simple events.

After training, we modify the complex event rules in the reasoning layer. We perform two experiments.

Experiment 1: Changing the Window Size In the first experiment, we make a small modification to the rules, reducing the window size to 2. Then, we evaluate the accuracy of the new system against the base testing dataset generated using a window of 2. This is the same dataset used for all the testing done in the sections above for systems that use a window of 2. Since we are using the perception layer that has already been trained, and that provides a good accuracy performance, we would expect that the system will be good at classifying the new complex events. Of course, we also expect the system to keep a good performance on the simple events as well.

In particular, as we are reducing the window size and thus simplifying the problem of detecting them, we would expect an increase on the performance when classifying the complex events.

This does not mean that we have learnt anything new due to the change, but simply that the system performs better since the problem is easier.

Experiment 2: Entirely New Rules For the second experiment, we use an entirely new definition for the complex events, shown in Listing 5.4. These new rules detect complex event $ce0$ when a 0 appears in the window and a 1 appears in the last position of the window, without a 9 appearing between them. $ce1$ is detected if a 1 appears in the window and a 2 appears in the last position without a 9 between them, and so on.

```
window(10).

happensAt(ce0, T) :- window(Window),
    sequence([0, exclude(9), 1], Window, T).
happensAt(ce1, T) :- window(Window),
    sequence([1, exclude(9), 2], Window, T).
happensAt(ce2, T) :- window(Window),
    sequence([2, exclude(9), 3], Window, T).
happensAt(ce3, T) :- window(Window),
    sequence([3, exclude(9), 4], Window, T).
happensAt(ce4, T) :- window(Window),
    sequence([4, exclude(9), 5], Window, T).
happensAt(ce5, T) :- window(Window),
    sequence([5, exclude(9), 6], Window, T).
happensAt(ce6, T) :- window(Window),
    sequence([6, exclude(9), 7], Window, T).
happensAt(ce7, T) :- window(Window),
    sequence([7, exclude(9), 8], Window, T).
```

Listing 5.4: New complex event rules for Experiment 2, to evaluate how DeepProbCEP performs after changing the complex event rule definitions.

Note that the following important points have changed:

- The number of complex event types has been reduced from 10 to 8.
- The window size has changed from 5 to 10. This is defined in the first line of Listing 5.4.
- The definition for each of the complex events is significantly different. Most notably, these definitions make use of the negated value, which indicates that a certain simple event does not occur between two other complex events. This was not used in the previous chapter as Neuroplex does not allow for this definition, and thus a fair comparison was not possible.

We have evaluated this by calculating the accuracy provided by the system for a testing dataset generated using the new complex event rules. This testing datasets has been generated in a similar manner to the base datasets described above in Section 4.5.1, but changing the conditions under which a complex event is generated to match the rules shown in Listing 5.4. For the purpose of demonstrating that the system allows the rules to be changed, we will again evaluate the accuracy for both the complex event and simple event classifications.

Results

Experiment 1: Changing the Window Size For the first experiment, after changing the window size and loading the trained weights into the perception layer, we evaluate the performance against the testing dataset for a window of 2. It is important to note that no further training is done. Instead, the weights resulting from training with the data-scarcity dataset are simply loaded into the neural network in the perception layer. As such, it is not surprising that the performance on the simple events remains the same, as we are effectively using the same neural network to perform the classification. This means that the accuracy for the simple events remains exactly the same, at 97.8%.

The performance when classifying the complex events, however, has changed to 99.1%, which is an increase of 1.6% when compared to the performance on a window of 5. As explained above, this is due to the simplification of the problem, as the system is more likely to get 2 out of 2 simple events correct than it is to get 5 out of 5. As such, this demonstrates that the DeepProbCEP approach allows users to make small modifications to the rules of the system. As long as the performance of the perception layer used performs well and the new rules are appropriate for the situations that the user is trying to detect, the system as a whole will also perform well.

Experiment 2: Entirely New Rules After updating the rules to the ones shown in Listing 5.4 and loading the trained weights into the perception layer, we have evaluated against the testing dataset generated with the new rules. Again, if we evaluate the performance on the simple events, this remains the same, as we are effectively using the same neural network.

More interestingly, however, an accuracy of 96.3% is archived on the complex events. Since the changes for this experiment are much more substantial than the previous one, this demonstrates that the reasoning layer can be changed as much as needed, without requiring any further training.

It is also worth noting that, the decrease of 1.2% in the performance of the complex events is due to the added complexity of the problem, in a similar manner to how the performance increased in the previous experiment. In particular, the increase of complexity is due to the fact that the window size has doubled and that there is an added `excluded` clause in the definition of the complex events.

As a result of these experiments, we are confident that changes to the reasoning layer are as easy to implement as changing the rules for the complex events, and loading the trained weights into the perception layer. These changes, however, are limited by the fact that the same simple event types must be used, as the perception layer must remain unchanged. In the following section, we explore how we can deal with cases in which the types of simple event need to be changed.

5.2.3 Changing the Perception Layer

As shown in the previous section, the DeepProbCEP approach allows for very significant changes to the perception layer. This can be useful to change the representation of the complex events very quickly and easily. There are cases, however, where we need to change how the perception layer identifies simple events. For example, this may happen if a new simple event type, which was not present in the original training data, is detected. Otherwise, this may happen if users identify that the classification for some of the simple events does not match their expectations. In these cases, we require further training data that matches the desired behaviour.

The easiest option is to re-train the perception layer from scratch with the new training data. However, a common reason why a type of simple events may have been missed is the fact that it is quite rare, hence, it may be hard to obtain enough training data to create a balanced dataset to train the dataset from scratch. Furthermore, as we have said, we want our approach to be agile.

This means that changes to the behaviour that can be implemented quickly may be preferable, as long as the accuracy is not significantly affected.

For the experiments in this section, we assume that we only have a small number of training data for the new type of simple event, and that it is not possible for us to wait until we collect more.

Experiment Design

In this section, we describe the different approaches we have evaluated in our experiments, as well as the experiments themselves. In order to simulate the situation in which a simple event was initially missing from the training data, we generated datasets based on MNIST digits. First, we generated a synthetic dataset that is missing all instances of one of the digits (in our experiments, this digit was 9). This used the same process as the base datasets, as described in Section 4.5.1 above. However, the fact that there are no instances of the digit 9 in the input stream means that $ce9$ will never occur. For our experiments, we balanced this dataset to have 500 instances for each of the nine complex event types that can still happen ($ce0$ to $ce8$), resulting on a dataset with 4,500 instances. This allows us to train the system to predict those nine complex event types with very high accuracy. We will refer to the system trained using only this dataset as the *pre-trained model*. This pre-trained model is very good at detecting instances of the complex events $ce0$ to $ce8$, but it is incapable of predicting instances as $ce9$. This is not only because it has not been trained to recognize instances as $ce9$, but because the perception layer does not even have an output corresponding to the MNIST digit 9, and there are no rules defining $ce9$ on the reasoning layer.

Then, we have generated a small number of training instances (10 instances) for the complex event $ce9$. These instances have been generated using all 10 classes of MNIST digits as input. The big difference in the number of instances for $ce9$ compared to the instances present for each class in the other dataset is meant to reflect the fact that $ce9$ is rare, and thus it is difficult to obtain training data for it. Specifically, we have 50 times more data for each of the other complex event types.

Next, we describe the different approaches compared in the experiment. These approaches aim to obtain good performances on the testing dataset, both for the overall accuracy and for the recall of the class $ce9$. This is because we want a system that is able to classify $ce9$ correctly. However, we do not want this to come at a significant cost to the accuracy of the other classes. For instance, classifying all instances as $ce9$ would give us a perfect recall, but the system would be entirely useless at detecting any of the other complex events, or even when no complex event occurs. On the other hand, using the *pre-trained model* would give us a decent overall accuracy, but the model would not be able to detect instances of $ce9$. As we want a system that performs well for all types of complex event, it is important that we look at both values.

The first approach that may come to mind would be to combine the two datasets, making a dataset with 4,510 training points that includes instances for all the complex event types. However, this dataset would be severely unbalanced, which would cause issues where the class with fewer instances would not be trained correctly. Two possible solutions would be *undersampling* or *oversampling*, and then training a new model using one of the resulting datasets. Using undersampling, we would get a training dataset with a small number of instances for each of the complex event types, which may not be able to train to a very good performance. Using oversampling would result in a bigger dataset. While the amount of training data for the new event type would still be the same, other complex events may perform better with more training data.

Another approach would be to make use of the weights from the *pre-trained model*. As described above, simply using the pre-trained model would not work well, as it would not be able to classify instances of $ce9$. However, we can make use of the perception layer weights and *fine-tune* them to allow it to identify instances of the digit 9. In our experiments, this has been done by simply changing the last linear layer of the model to have an additional output. This means that all the weights for the model can be loaded and re-used except for this last layer, which is randomly initiated instead. Of course, the definition for $ce9$ also needs to be added to the reasoning layer, which can be done by simply adding the corresponding rule to the complex event definitions. This means that the system has already been mostly trained, and should only require a relatively small amount of further training to learn the new class. For comparison purposes, we have performed this fine-tuning using both the undersampled and oversampled datasets.

Finally, something else to take into account is the fact that part of the neural network can be *frozen*, meaning that the weights are not changed anymore with further training. Of course, this does not make sense when training from scratch, as we want to train the weights for the whole neural network. However, when performing fine-tuning, the weights for part of the neural network may already be trained enough. As such, freezing those weights can reduce the time required to further train the system without reducing the performance afterwards, or possibly even increasing it. In the case of our perception layer, it is a neural network designed to classify MNIST digits. This neural network consists of an encoder and a classifier. The encoder is formed by two sets of a Convolutional layer and a MaxPool layer. Meanwhile, the classifier is composed of three fully connected linear layers. While the weights for the classifier may change more due to the addition of a new class, the weights for the encoder may not need to change to give good accuracy. In our experiments, we also evaluate how freezing the encoder affects the performance, using both the undersampled and the oversampled datasets.

Table 5.3: List of the different approaches used to address adding a simple event.

Approach	Dataset Used	Loading Weights	Frozen Encoder
Pre-trained Model	None	Yes	N/A
Training on Undersampled	Undersampled	No	No
Fine-tuning on Undersampled	Undersampled	Yes	No
Fine-tuning on Undersampled with Freezing	Undersampled	Yes	Yes
Training on Oversampled	Oversampled	No	No
Fine-tuning on Oversampled	Oversampled	Yes	No
Fine-tuning on Oversampled with Freezing	Oversampled	Yes	Yes

This results in 9 different approaches, as summarized in Table 5.3. First, we consider the approach of using the pre-trained model, which does not consider the new simple event type at all. Then, we consider both training from scratch, fine-tuning and fine-tuning with the encoder frozen. For these three approaches, we evaluate how they perform both using the undersampled and oversampled datasets.

In order to compare the performance of these approaches, we evaluate: (i) the accuracy when classifying the complex events, (ii) the accuracy when classifying the simple events and (iii) the amount of time required to train the system. As the accuracy when classifying complex events depends on the accuracy for classifying complex events, we discuss them together. Of course, we want as high of an accuracy as possible. However, as discussed above, in some situations the system may need to be updated quickly. In those cases, it may be preferable to have slightly lower accuracy, but a shorter training time. For this purpose, we also compare how long each of the approaches requires to train. As the fine-tuning approaches start with a mostly trained neural network, we would expect them to converge quicker than training from scratch, making them faster to train.

Accuracy Results

In this section, we show and discuss the accuracy performances provided by each of the approaches. The same testing dataset has been used for all approaches. This test dataset contains a similar amount of simple events for each of the types. While this does not necessarily reflect the scenario proposed above where the new event type is rarer, it will make it easier for us to detect when the performance classifying that simple event is lower. This is because, if the new simple event was indeed rarer in the test dataset, the overall performance would not change as significantly if the system was classifying those cases incorrectly.

In Table 5.4, we can see the results for each of the different approaches. As the table shows, using fine-tuning with the undersampled dataset performs much better than training from scratch. This could have been expected for the overall accuracy, as the system was already able to perform fairly well for most classes. However, the recall for the new event type shows that the system also performs better in that case. This is particularly true for the case in which we freeze the encoding layer, which seems to perform better than not freezing it in terms of the new event type.

However, quite interestingly, the fine-tuning approaches seem to perform significantly worse at classifying the new event type after training with the oversampled dataset. This may be caused by an over-fitting in that class, which can be an issue when using oversampling. In contrast, the approach of training from scratch seems to perform much better when using this dataset. This is likely thanks to the larger amount of training data for all the other classes, which allows this approach to obtain the best overall performance for both complex and simple events. However, it is worth noting that the performance using the fine-tuning approach with the undersampled dataset is almost the same, while giving much better results in terms of the recall for the new class.

The table also shows the results of using the pre-trained weights without any fine-tuning. In this case, the overall accuracy is still fairly good. However, as expected, the recalls for the new simple event class are at 0, as it is impossible for this model to predict the new event type.

Finally, the table also shows results for training with 1,000 and 5,000 samples. These cases show the results of training with balanced datasets in which we have more training data for the new event type. As expected, these cases perform significantly better than any of the previous approaches.

Table 5.4: Results when adding simple event. Best results indicated in bold. Results below the horizontal line are provided for comparison purposes, but they do not follow the restrictions applied to the other approaches.

Approach	Complex Events		Single Digit	
	Accuracy	Recall	Accuracy	Recall
Pre-trained Model	0.8970	0.0000	0.8853	0.0000
Training on Undersampled	0.8468	0.7415	0.8518	0.8371
Fine-tuning on Undersampled	0.9562	0.7756	0.9616	0.8679
Fine-tuning on Undersampled with Freezing	0.9575	**0.8049**	0.9620	**0.8821**
Training on Oversampled	**0.9599**	0.7017	**0.9653**	0.8226
Fine-tuning on Oversampled	0.9587	0.6638	0.9641	0.7988
Fine-tuning on Oversampled with Freezing	0.9592	0.6420	0.9639	0.7826
Training with 1,000 samples	0.9637	0.9119	0.9681	0.9501
Training with 5,000 samples	0.9755	0.9318	0.9785	0.9680

Training Time Efficiency

While accuracy performance is quite important, shorter training times may allow us to update the system faster, which may be preferable in some cases. In Table 5.5, we show results indicating how many epochs each approach requires to train, as well as how much time the training can take in total. The results are obtained from averaging three different executions, which causes the epoch number to not be an integer in some of the cases. It is worth noting that the number of epochs can vary because we use an early stopping approach with patience of 10. This means that if the best accuracy on the validation dataset does not improve for 10 epochs, the training is stopped and the best weights are used.

In Table 5.5, we can see that approaches that use the oversampled dataset take much longer per epoch than the ones using the undersampled dataset. Of course, this is because there are

Table 5.5: Training time-efficiency for the different approaches. Best in bold.

Approach	Time per Epoch (s)	Average Epochs	Total Training Time (s)
Training on Undersampled	2.836	52.0	147.476
Fine-tuning on Undersampled	2.819	30.7	86.458
Fine-tuning on Undersampled with Freezing	**1.755**	35.0	**61.425**
Training on Oversampled	137.287	22.7	3111.843
Fine-tuning on Oversampled	138.466	**12.3**	1707.745
Fine-tuning on Oversampled with Freezing	83.850	17.3	1453.407

50 times more samples in the oversampled dataset, making it much slower to process all the samples in the dataset. We can also observe that, as long as the encoder layer is not frozen, training from scratch and fine-tuning do not differ significantly in terms of the time needed per epoch. Both of them train the whole neural network, and having loaded weights at the start does not make any difference in this regard. However, freezing the encoder layer does result in a significant increase in the speed at which each epoch is performed. This is because fewer parts of the neural network need to be updated and, as such, it is much faster to calculate the weights.

In terms of the number of epochs needed to train, we can see that the fine-tuning approaches take significantly fewer epochs to converge than the training from scratch approach. This could have been expected, as the loaded weights should already be quite close to the optimal weights for the problem. This is because they were already trained to classify a very similar problem, as we have simply added a class to the output. Similarly, the approaches that use the oversampled also require a smaller number of epochs. Most likely this is because the higher number of samples in the oversampled dataset provides much more training data per epoch, thus reducing the number of epochs needed. On the other hand, however, this also increases the amount of time needed to process each epoch, resulting in a longer total training time.

Interestingly, the approach that does not freeze the encoder layer requires fewer epochs to converge than the approach that does freeze it. Being able to modify more of the weights in the model for each sample most likely allows it to converge quicker. However, freezing the encoder significantly reduces the amount of time needed per epoch. As a result, the approach of freezing the encoder is the fastest with either dataset in terms of total training time.

Therefore, if the model needs to be updated quickly, the approach of fine-tuning with a frozen encoding layer using an undersampled dataset may be the best choice, as it results in the system with the best recall for the added complex event and digit classes, while being the fastest approach by far. At the same time, while training from scratch on the oversampled dataset produces the best overall accuracy for both complex events and single digits, the fine-tuning approach is not far off. These specifics, however, may change depending on each specific situation. As such, further research on the benefits of each approach may be needed before reaching a definitive conclusion.

5.2.4 Using Audio as the Input

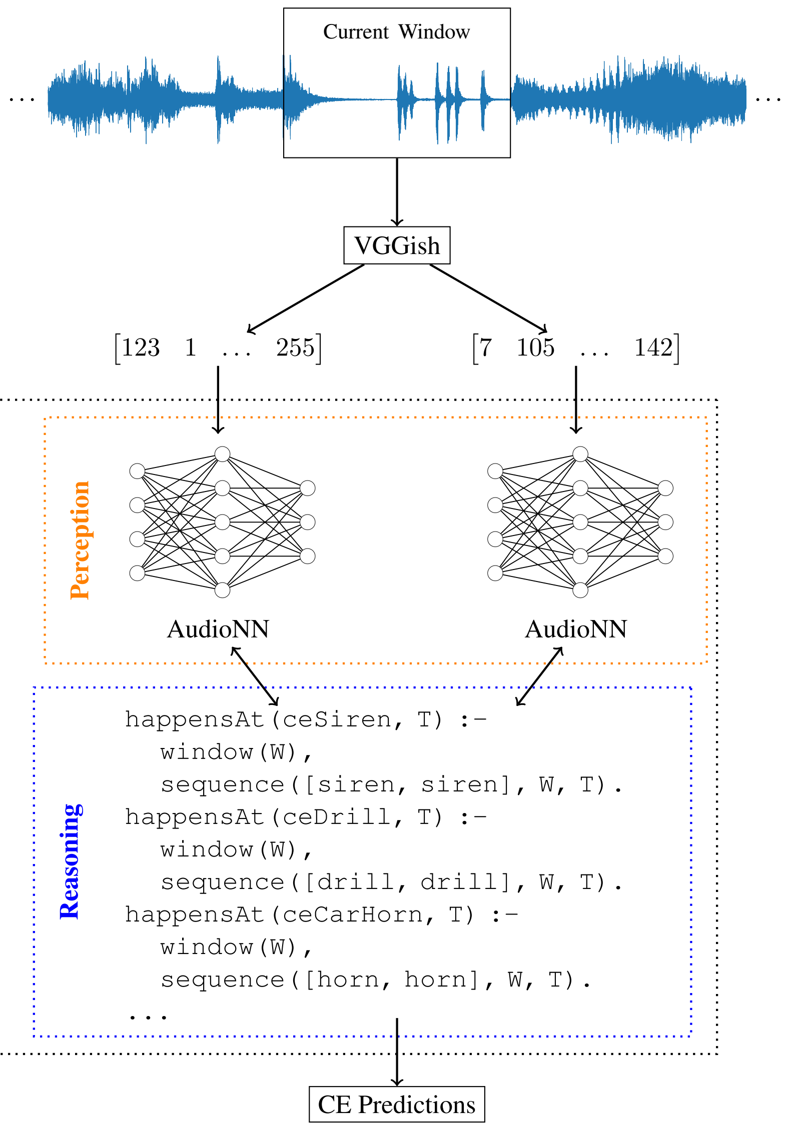
Another way in which the DeepProbCEP approach is designed to be agile is by allowing multiple types of input data. In this section, we demonstrate this by implementing a system that detects complex events from an audio feed. This not only demonstrates that our approach can work with multiple input types, but is also a more realistic scenario, using types of data that could be directly used from input sources in the real world.

In the following sections, we explain how this system has been designed to deal with audio, as well as describing in which ways it differs from the system implemented for the MNIST case. We also evaluate the accuracy of this audio system by training it using a synthetic dataset generated based on urban sounds.

Audio System

Since both approaches use the DeepProbCEP approach, the architectures for processing audio and MNIST digits are quite similar. The architecture used for the audio case is shown in Figure 5.13. First of all, the input data is pre-processed. In this case, we have a stream of audio as an input, which is divided into 1-second segments by VGGish. VGGish is a state-of-the-art feature extractor for audio classification models.[1] VGGish performs a feature extraction process which results in a matrix of size $128 \times N$, where N is the length of the input audio file in seconds. Each position in the matrix contains a value between 1 and 255. For our approach,

[1]In order to make it compatible with DeepProbLog, we use a PyTorch implementation of VGGish, available at `https://github.com/harritaylor/torchvggish`



```
happensAt(ceSiren, T) :-
  window(W),
  sequence([siren, siren], W, T).
happensAt(ceDrill, T) :-
  window(W),
  sequence([drill, drill], W, T).
happensAt(ceCarHorn, T) :-
  window(W),
  sequence([horn, horn], W, T).
...
```

Figure 5.13: Architecture of the system used for the audio scenario.

we consider each 1-second segment to be a simple event, meaning that each timestamp refers to the corresponding 1-second long segment of audio.

As in the MNIST case, the matrix resulting from the pre-processing is fed into a neural network. In this case, we use AudioNN, which consists of a multilayer perceptron (MLP) that has 5 layers with 100, 80, 50, 25 and 10 neurons, in this order. A ReLU activation function is used between each of the layers, and a Softmax activation function is applied at the end. This allows us to classify the input audio into one of 10 classes, which is the number of classes in the audio dataset we will be using. Finally, the output of AudioNN is used in the complex event definitions. For the purposes of evaluating the accuracy, we define the rules in a similar manner to how they were defined for the MNIST case, where two instances of the same class happening within the window size results in a complex event. Figure 5.13 also shows a snippet of the code used for detecting complex events.

Audio Dataset

The audio datasets are generated in a similar manner to the MNIST dataset, explained above in Section 4.5.1. However, instead of using MNIST as the original dataset, Urban Sounds 8K [Salamon et al., 2014] is used instead. Urban Sounds 8K is a dataset that contains short audio files (up to 4 seconds in length) from 10 different classes that can be found in an urban setting: air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, and street_music. Urban Sounds 8K has pre-sorted the audio files into 10 folds.

As both the MNIST and Urban Sounds 8K datasets have 10 different classes, each of the classes in the audio dataset substitutes one of the digit classes. With this, the steps described in Section 4.5.1 are followed to generate the training, validation and testing datasets. As a reminder, this process generates a sequence containing all the instances from the original dataset, and then detects where complex events occur according to the rule definitions. Folds 1 to 8 are used to generate the training, fold 9 is used to generate the validation dataset and fold 10 is used for testing. It is also worth noting that, for simplicity and in order to use consistent audio lengths for each of the files, only the first second of each of the audio files is used. Audio files with a length shorter than one second are discarded. As with MNIST, we have generated datasets with different window sizes, ranging from 2 to 5. In this case, the window size corresponds to the window of time in seconds within which we are looking for the given pattern. Finally, the training datasets are balanced and given a fixed size of 1000 training points to allow for a fair comparison between window sizes.

In order to evaluate the performance of the audio system, we have trained it using the datasets for each of the window sizes and then evaluated its accuracy on the testing datasets of the corresponding window size. When training this system we have used a maximum number of epochs of 100, with early stopping if the accuracy does not improve for 10 epochs.

Results

Table 5.6: Average accuracy results and standard deviation for DeepProbCEP in complex events classification and classification of sounds of 1 second in length.

Window size	**Complex Event**		**Individual Sound**	
	Accuracy	**STD**	**Accuracy**	**STD**
2	0.8657	0.0041	0.6696	0.0100
3	0.7645	0.0109	0.6497	0.0265
4	0.7069	0.0191	0.6501	0.0284
5	0.6401	0.0225	0.6410	0.0694

In Table 5.6 we can see the results of training our approach with the datasets of different window sizes. As shown in the table, the performance of the approach is fairly good with a window size of 2. However, the performance does decrease as the window size increases. This could have been expected, as the problem gets more complex as the window size increases. This is because a bigger window size contains more simple events, which makes it more likely that the system will incorrectly classify one of them. This can cause the system to predict that a complex event is happening when it is not, thus reducing the performance of the system. Interestingly, however, the accuracy for classifying individual sounds (that is, a 1-second long segment) is fairly consistent over the different window sizes.

While this system performs worse than the one used for MNIST did in similar circumstances, this could have been expected, as the problem of classifying audio is a harder task than classifying MNIST digits. The accuracy for classifying individual sounds of our approach is comparable, albeit slightly lower, to the performances obtained by several approaches to classifying the sounds in Urban Sounds 8K. It is also worth noting that these approaches train using Urban Sounds 8K directly, meaning that they use all the training data from the dataset. In comparison, our approach only uses 1 second from each of the files, meaning that it has less training data and less information in the test cases. Furthermore, only the subset of files that generate a complex event are used for training, meaning that even less data is used to train the system. Finally, the labels provided to the system for training indicate when the complex events happen. This means that, effectively, we are only telling the system that two out of all the 1-second segments in the window are of a certain class. As such, we would argue that the problem of learning to classify sounds in the context of CEP is clearly a harder problem than learning them from training with Urban Sounds 8K directly. Therefore, we believe that obtaining a performance that is only slightly lower demonstrates that our approach is capable of learning to classify sounds correctly.

We have also performed the same robustness experiments as we performed on the MNIST case in Section 5.1. The results for the audio case, which have been added in Appendix C, confirm what we saw when evaluating the results for the MNIST case. The main difference is the fact that, due to the increased difficulty of the audio problem, the performance is lower than in the MNIST case, and is more affected by higher levels of noise. Despite this, the accuracy of the system is not significantly affected when training with datasets with 20% of noisy data.

As a result, we believe that the DeepProbCEP approach can be useful for detecting from streams of data of many different types, as long as the structure for the perception layer is chosen appropriately.

5.3 Discussion and Conclusion

5.3.1 Robustness

As demonstrated in this chapter, DeepProbCEP is highly robust against many types of adversarial poisoning attacks. DeepProbCEP is able to maintain its accuracy when 20% of the data is noisy in all the attacks we have considered, and even maintaing the accuracy when over 50% of the data is noisy in most cases. More specifically, DeepProbCEP is robust against both attacks that targeted all the classes, as well as an individual class. However, attacks against a single class may not be noticeable when evaluating the performance of the system as a whole, which means that evaluating the performance of each individual class is necessary if we want to ensure that the system is performing well for all of them.

Limitations in robustness

Despite DeepProbCEP proving to be quite robust against the poisoning attacks in the MNIST scenario, it is worth considering that it may not perform quite as well in more difficult problems. For instance, when looking at the same poisoning attacks from Section 5.1 using the audio setting (results shown in Appendix C), we found that the accuracy was not affected when 20% of the data was noisy, but higher percentages of noise did seem to have more of an effect. As such, it seems reasonable to expect that for more complex problems the percentage of noise needed to have a significant effect will be lower.

It is also worth noting that other types of noise could be considered in the future. For instance, depending on the definition for the complex events, the noisy labels could be maliciously changed in a way designed to perjudicate the training of just one simple event. For instance, if complex event $ceSG$ is defined as a siren followed by a gunshot, a malicious agent could incorrectly label sirens followed by other sounds as $ceSG$. While this may not have any significant effect on the performance of the neural network when classifying a siren, it may significantly harm the ability of identifying gunshots, which could have unexpected consequences for the detection of the complex events. We would not necessarily expect these effects to be larger than the ones shown in this chapter, as it would likely only effect the targeted complex events, or at most the complex events that make use of the targeted simple event (depending on the specific rules and poisoning process). However, the fact that only one simple event is targeted may make it harder to detect. Furthermore, even if users detect that the poor system performance is due to the perception layer being unable to correctly identify one of the simple events, they may simply assume this is due to that simple event being harder to classify.

Finally, it is also worth considering that, depending on the type of input data, there are many other types of attacks that could be performed against the system. For instance, if using video data such as in Chapter 3, malicious agents could modify the training videos by lowering the quality of the video, or adding artifacts that make it harder to identify what is happening. There are also attacks that can be performed on the system after it has been trained, exploiting imperfections in its classification. For instance, in [Thys et al., 2019] authors show that object detectors (and more specifically, person detectors) can be avoided by holding an adversarial patch, which prevents the system from detecting the person behind it. Clearly, this type of approach could be used against systems that rely on such object detectors, and similar versions could be used for any type of input data we wish to use DeepProbCEP on. As such, while we have shown DeepProbCEP to be robust against some types of attacks, we should still consider that there may be other situations against which it is not robust.

5.3.2 Agility

This chapter also shows the agility offered by the DeepProbCEP approach. This is offered by a combination of features that allow the user to change the behaviour of the system easily and quickly in any situation. We have explained how frameworks can be used to define complex events in different manners. We have also shown that the behaviour of a system can be changed after training, either by modifying the reasoning layer or by fine-tuning the perception layer. Finally, we have also shown how DeepProbCEP can be applied to detect complex events from other types of input data, considering the use of an audio stream.

Limitations in agility

As described above, the limitations on changing the behaviour of the reasoning layer are very small, allowing for any change as long as the interaction with the perception layer remains unchanged. On the other hand, the usefulness of the fine-tuning approach is significantly more limited. Indeed, the accuracy of the approach can be slightly worse than training from scratch. This difference is likely to be even more significant in situations where users want the perception layer to have a more significant change in behaviour. As such, the fine-tuning approach is likely to only be useful in cases where users need to make a slight change to the behaviour of the perception layer in a short time period. Despite this, we believe that in some situations, the significant reduction in the time required to train the system may outweigh the slight decrease in performance of the approach.

Chapter 6

Discussion and Conclusion

In this section, we summarize the main achievements described in this thesis, and describe how they relate to the objectives described in Chapter 1. We also consider again the motivating scenario discussed in Section 1.1, and explain how our achievements led to the implementation of that demonstration. Finally, we propose different areas of potential future research.

6.1 Advances in Research

In this section, we discuss the main achievements and consider how they relate to our three main objectives from Chapter 1. As a reminder, these are (i) **agility**, or the ability to change quickly and easily, (ii) **reliability**, or the ability to function as expected under a wide range of operational conditions, and (iii) **efficiency**, or the ability to perform tasks with good use of time and training data.

In Chapter 3, we present ProbCEP, an approach that allows users to combine pre-trained proxy models to detect complex events from feeds of data. We demonstrate that, when combined with human knowledge, this can be used to detect violence from CCTV feeds with reasonable accuracy. This approach focuses on providing **agility**, allowing users to quickly deploy a system without the need for training data. Of course, the use of proxy models trained on unrelated data can make it difficult to obtain great accuracy results, but the ease of implementing new systems can allow users to deploy a functioning system in situations where other approaches are unfeasible.

Chapter 3 also presents the PreCompilation approach, which allows users to significantly speed up the rate at which ProbLog queries can be performed, as long as the format of the query remains consistent. Using PreCompilation, the ProbCEP approach can be implemented in a **time-efficient** manner.

In Chapter 4, we compare different approaches to non-symbolic-focused CEP in terms of their **data-efficiency**. We demonstrate that neuro-symbolic approaches are significantly more **data-efficient** than neural-only approaches, making neuro-symbolic approaches more suited to problems where data is sparse. Towards the end of the chapter, we also explore the **reliability** of the different approaches, both in terms of consistency in the accuracy performance and how much risk they present on behaving in a manner that we do not expect. In this regard, we show how DeepProbCEP results in the most consistent accuracy results, both in terms of separate executions and over a wide range of window and training dataset sizes. We also explain how DeepProbCEP's implementation of the reasoning layer ensures that it will exactly follow the rules we have defined, whereas other approaches are unable to ensure this.

Finally, Chapter 5 is divided into two main sections. Firstly, we demonstrate that DeepProbCEP is robust against multiple poisoning adversarial attacks, providing further evidence of a **reliable** behaviour. In the second half of the chapter, we also explore which features allow DeepProbCEP to be an **agile** approach. This is done by allowing flexibility to users through the use of frameworks, which allow them to define complex events in the manner that best matches how these actually happen, as well as allowing users to change the functionality of the system after training, both at the reasoning and perception layer. Chapter 5 also shows that DeepProbCEP can be used to process different types of input data, demonstrating this in the audio setting. Furthermore, this also demonstrates that DeepProbCEP can be used in a scenario that is much closer to the real world, thus showing the potential for the DeepProbCEP approach on real data. While some of the experiments shown in this thesis are using simpler datasets like MNIST, the fact that DeepProbCEP can perform fairly well in the more realistic audio scenario shows that the results should be transferable to more complex situations that have not been considered in this thesis.

In the following section, we demonstrate how most of these advancements can be combined to detect complex events from a real-world audio recording.

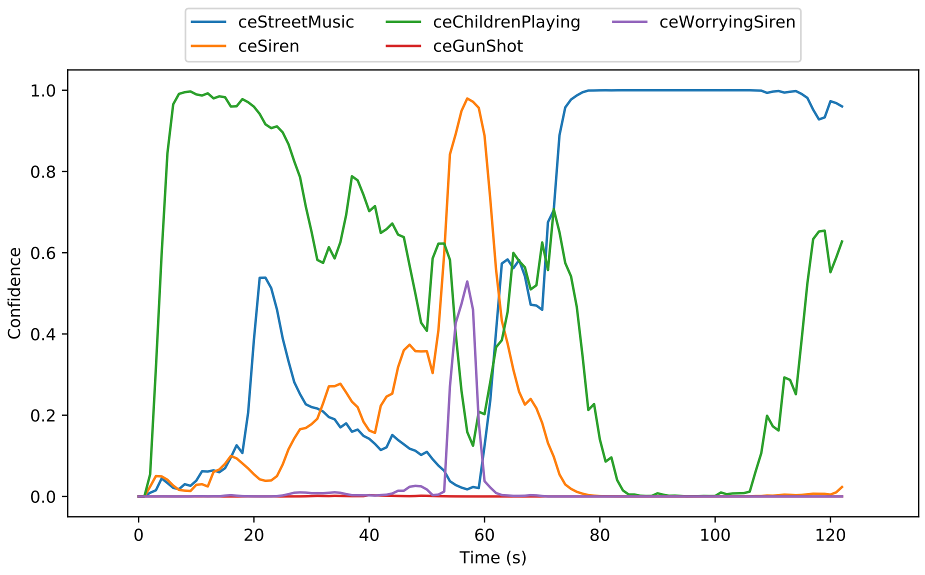
6.2 A Real-World Example

Let us recall the example from Chapter 1 showing how a system was able to detect several complex events from an audio feed. In particular, the system was able to detect when the complex event *worrying siren* was happening. This was defined as a complex event where a siren was heard in a worrying situation. In this section, we will go over how this was defined using rules, as well as all the other steps required to obtain these results. In the video[1], two different audio files are given to the system. The first audio file is a recording taken by someone crossing a busy street, which starts with background voices, continues with the sound of sirens and finishes with street music. The second audio file starts with the same recording, but as the sound of sirens ends, a series of gunshots has been edited in, substituting the street music. The system is able to identify that the second video contains a *worrying siren*, as the sound of gunshots increases its concern about the situation. In contrast, in the first audio the confidence of the system that a *worrying siren* is happening quickly decreases when the sound of street music is detected instead.

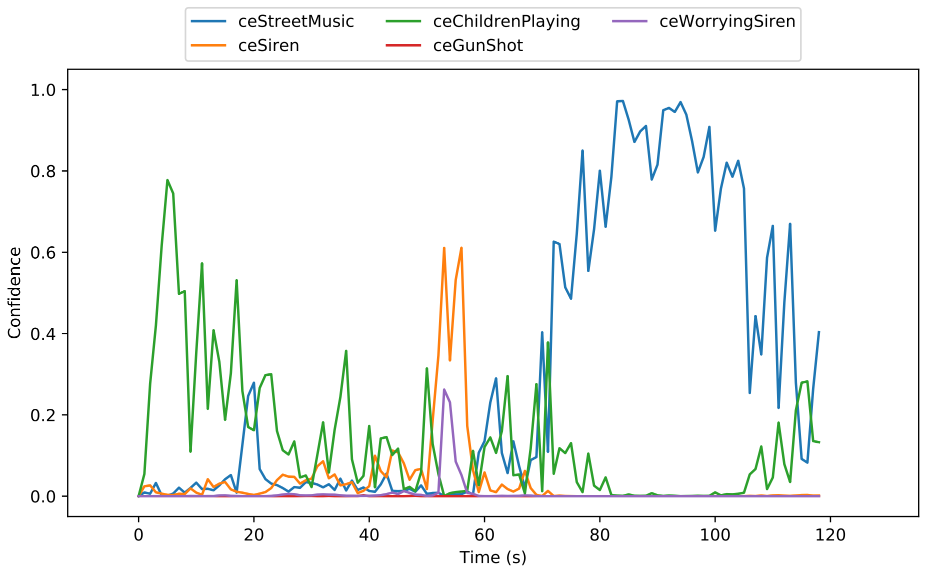
The first step towards implementing this demonstration was training the audio system to allow it to use audio as an input. For that purpose, we used the system described above in Section 5.2.4. For the purpose of this demo, the weights resulting from training with a window of 2 were used. The reason why we decided to use the training weights from this case, instead of training specifically for this problem was the fact that generating data for the case of the *worrying siren* would have been difficult due to the complexity of the definition for the complex event. Instead, we modified the rules in the reasoning layer to redefine the complex events as appropriate. First, we added the event calculus-inspired framework to allow complex events to last for an indefinite length of time, based on the initiation and termination rules. This allows us to define the confidence for each timestamp based on the confidence from the previous timestamp. As shown in Figure 6.1a, this produces less sudden changes to the confidence for the complex events, where the confidence only decreases quickly if we are confident that the complex event has finished according to the rules. In contrast, Figure 6.1b shows the result of using only the sequence framework to define the complex events, where the confidence can be

[1]As a reminder, the video demonstrating this system can be found at `https://www.youtube.com/watch?v=dl1H0VzppPM` (on 13th of April 2022)

reduced suddenly if the perception neural network stops being confident in the predicted class for a few seconds.



(a) Using event calculus inspired approach



(b) Only using sequence to define the complex events

Figure 6.1: Comparison of the output graph for the same file using an event calculus inspired definition of complex events and just using sequence.

For the purpose of this demonstration, we have also changed the window size to 5, instead of the window of 2 used for training. This new window size of 5 is defined in the code from Listing 6.1, which also shows some of the code for the `ceSiren` complex event. `ceSiren` is initiated when two instances of a siren are detected within the window, and terminated when no sirens are detected for 5 seconds. The rest of the complex events, excluding `ceWorryingSiren`, are defined in the same manner for the corresponding simple events.

```
givenWindow(5).

initiatedAt(ceSiren = true, T) :-
    givenWindow(Window),
    sequence([siren, siren], Window, T).
initiatedAt(ceSiren = false, T) :-
    givenWindow(Window),
    sequence([exclude(siren)], Window, T).
```

Listing 6.1: Code used to detect complex events for a single type of simple events.

The code in Listing 6.2 shows the code defining `ceWorryingSiren` based on the simple events that are occurring. In order to define this, we first define what types of sounds constitute a "safe environment". In lines 1 and 2, we define that children playing and street music are sounds that indicate that the environment is safe. Similarly, lines 3 and 4 define that gunshots and sirens are sounds that would increase how worried we are about the situation.

Following that, Listing 6.2 defines how the complex event can be initiated. First, lines 6 to 18 define that a `ceWorryingSiren` will be initiated if we start hearing sounds from a "safe environment", followed by at least 3 seconds of sirens and no more sounds from a safe environment.

After the worrying siren complex event has been initiated, we can further increase our confidence if we detect more "worrying sounds", without detecting sounds from a safe environment. This is defined in lines 19 to 24. More specifically, line 22 checks that a worrying siren complex event has already been started at the previous timestamp. Line 23 checks that no safe environment sounds are heard, and line 24 checks if multiple instances of worrying sounds are detected.

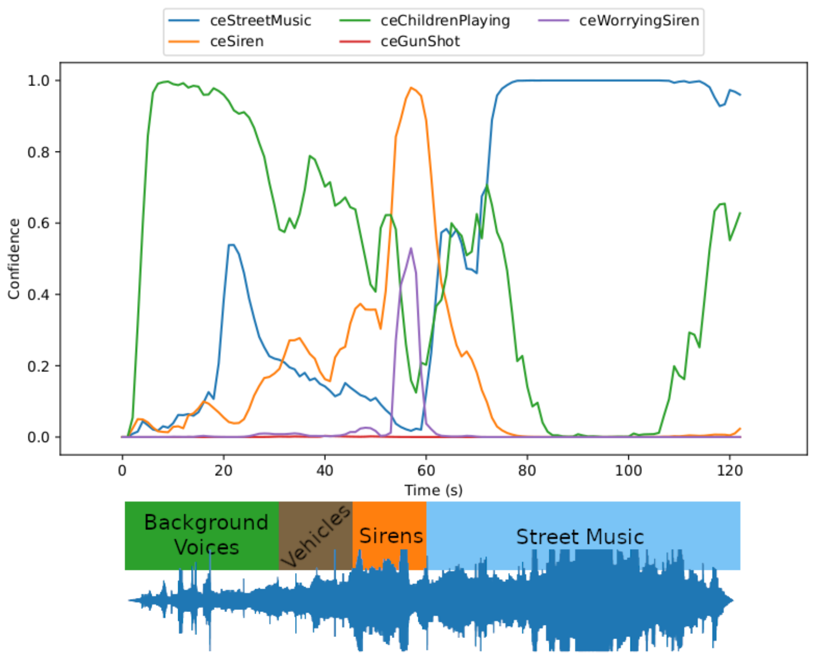
Finally, the last two rules terminate *ceWorryingSiren*. This happens if we stop hearing worrying sounds, or if we start hearing sounds from a safe environment. The first condition is defined in lines 26 to 28, which specifies that if no worrying sounds are detected, `ceWorryingSiren` should be terminated. Meanwhile, lines 29 to 31 define that if we start hearing safe environment sounds the complex event ends as well.

```
wraps(safeEnv, childrenPlaying).
wraps(safeEnv, streetMusic).
wraps(worryingSound, gunShot).
wraps(worryingSound, siren).

initiatedAt(ceWorryingSiren = true, T) :-
    givenWindow(Window),
    sequence(
        [
            safeEnv,
            siren,
            exclude(safeEnv),
            siren,
            exclude(safeEnv),
            siren,
            exclude(safeEnv)
        ], Window, T
    ).
initiatedAt(ceWorryingSiren = true, T) :-
    givenWindow(Window),
    allTimeStamps(Times), previousTimeStamp(T, Times, Tprev),
    holdsAt_(ceWorryingSiren = true, Tprev),
    sequence([exclude(safeEnv)], Window, T),
    sequence([worryingSound, worryingSound], Window, T).

initiatedAt(ceWorryingSiren = false, T) :-
    givenWindow(Window),
    sequence([exclude(worryingSound)], Window, T).
initiatedAt(ceWorryingSiren = false, T) :-
    givenWindow(Window),
    sequence([safeEnv], Window, T).
```

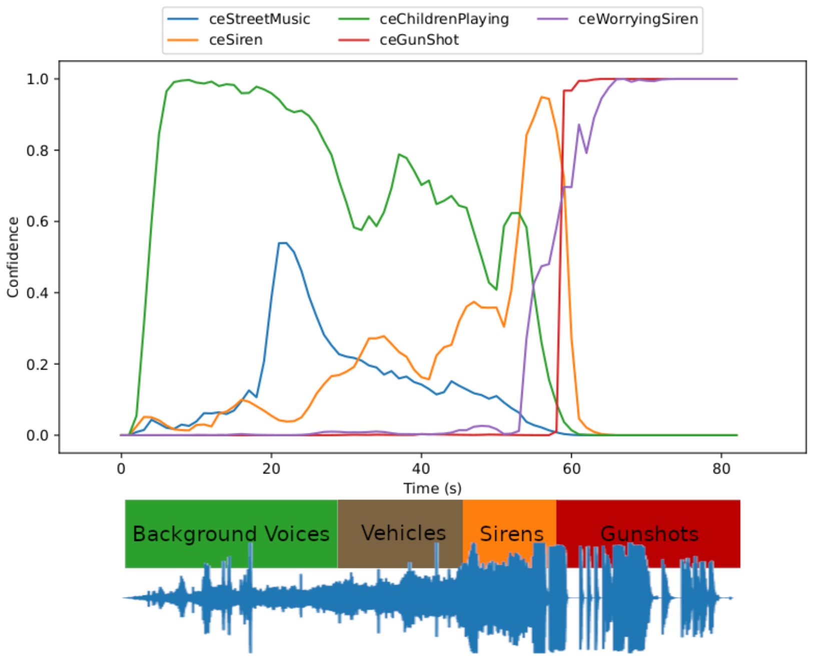
Listing 6.2: Code used to detect `ceWorryingSiren` in the demonstration.

As shown in Figure 6.2, these rules detect when a worrying siren is heard, based on the description above. In order to evaluate live inputs, the implementation of LiveEvents from Section 3.7.2 was extended to allow the use of audio as an input. While LiveEvents was originally designed to work with ProbCEP, it is also capable of using the neural networks that have been trained using a DeepProbCEP approach, effectively treating them as proxy models. When implementing the audio case in LiveEvents, we also defined which are the inputs and outputs for this problem, thus allowing us to use the PreCompilation approach when detecting these complex events. After these steps, the system can be used to detect the complex events from any given audio file, as well as audio from the computer's microphone.

A similar approach could also be used to perform video processing. Depending on the complex events that need to be detected, and what simple events they are based on, this could be approached in two ways: (i) treating each frame individually or (ii) looking at short fragments



(a) Original audio file



(b) Modified audio file with gunshots

Figure 6.2: Predictions for two audio files. The boxes between the waveforms and the graphs show the main sound present in the audio for each period of time.

of video. The first approach is closer to the process used in the MNIST scenario, where each image of the video would be treated individually and the simple events detected in each image would then be combined in the reasoning layer. This approach can be easier to train, but it may be harder to identify some actions where the movement is important.

The approach of using short fragments of video would be closer to the one used in Chapter 3, but training the neural networks instead of relying on pre-trained proxy models. This has the advantage of allowing users to consider simple events that need a few frames to be identified. However, DeepProbLog does have some limitations in which neural networks it is able to train. For instance, it is only able to output a pre-defined set of classes, meaning that it cannot train a neural network to output a number. This means that, among others, it is not able to train an object detector neural network, as it cannot effectively output the position of the objects. As such, some future work on DeepProbLog would likely be required to allow it to train for the scenario proposed in Chapter 3. This could partially be solved using an approach as the one proposed in [Apriceno et al., 2021]. In this paper, the authors treat a video of two MNIST digits moving towards each other on a black background until they meet. At that point, the MNIST digits may combine by adding or subtracting from each other, transforming into a single digit which is the result of the operation (see Section 2.3.3 for a more detailed explanation). The authors of [Apriceno et al., 2021] combine a pre-trained object detector that identifies the position of the MNIST digits in each frame with an MNIST classifier that identifies the value of those digits after they have been located. This approach could be expanded to deal with videos that show actions from the real world, which would likely be more complex to identify.

6.3 Future Work

While we believe that the approaches we have proposed in this thesis have a lot of potential when detecting complex events from non-symbolic data, these approaches do have some limitations, as explained at the end of each chapter. Solving this limitations opens up new research avenues, which we believe would extend the potential of the discussed approaches even further. One of the main limitations, both in ProbCEP and DeepProbCEP, is the fact that the rules need to be manually defined. Furthermore, they need to be defined accurately in order to be useful for detecting the complex events and, in the case of DeepProbCEP, training the system. This means that, in situations where it is not possible for experts to define such rules, these approaches cannot be used.

An area of research that could help to minimize this problem would be to learn the rules automatically, in a similar manner to the approaches discussed in Section 2.2.1. The CEP approaches discussed in that section are able to automatically learn complex event rules similar to those defined by humans based on training data. However, existing rule-learning approaches are not able to deal with non-symbolic data. An area of research that we believe would have a lot of potential solving this would be combining these rule-learning systems with the approaches proposed in this thesis. This would require integrating the process of learning the rules with the process of extracting symbolic information from the non-symbolic data. This could improve the accuracy of the rules, thus improving the performance of the system as a whole. In particular, this could allow approaches like ProbCEP to make better use of their proxy models. Based on some early experiments, we also believe that DeepProbCEP could make use of a rule-learning approach. As discussed in Section 4.9.1, we have done some preliminary research using FastLAS [Law et al., 2020, Law et al., 2021], an inductive logic programming approach, to learn the complex event rules. In [Preece et al., 2021] we discus how this approach was used to create a demonstration where the rules learnt by FastLAS were successfully used in DeepProbCEP to detect complex events. However, further experiments are required to evaluate the full extent of capabilities for this type of approach.

Another possible solution to the problem of learning the rules are the solutions introduced in [Dai and Muggleton, 2021, Cunnington et al., 2022]. These papers introduce a new area of research where neuro-symbolic approaches that are able to simultaneously learn symbolic rules and train a neural network. While this field of research is still in early stages, and is yet to be applied to CEP problems, we believe that it has great potential for this type of circumstances. In particular, it could allow us to train in circumstances where proxy models are unavailable (making it impossible to use approaches such as ProbCEP) and where experts are unable to define the required rules accurately (preventing the use of neuro-symbolic approaches). However, further research is necessary to implement such approaches in the field of CEP.

It would also be interesting, for either of those rule-learning solutions, to compare the performance of automatically generated rules with those given by experts. While we have demonstrated that injecting accurate human knowledge into the system can improve its accuracy and ability to train with sparse data, in some cases experts may define slightly incorrect rules. This would likely diminish the accuracy of the system. However, without further research it is unclear to what extent the performance of the system would be affected based on how inaccurate the rules are, and how this would compare with a rule-learning approach.

Another limitation of the proposed approaches is the fact that they are only designed to work in a centralized manner, and have only been tested with a single input stream. There is potential for new research avenues trying to solve these limitations, considering distributed CEP approaches or CEP systems that allow for multiple input types at the same time. For instance, an approach could be designed that allows the combination of multiple inputs types (such as the images and sound from a video) to provide more accurate results. This could go further, including the consideration of multiple sensors, distributed over an area, being able to detect the complex events that are happening and their position within that area. This however, also introduces a number of difficulties, depending on the implementation details. For instance, it opens the possibility of synchronization issues, as different sensors may keep track of time in different manners. Another issue could be the hardware performance, as it might not be able to keep up with a large number of different input streams. Furthermore, while DeepProbLog is theoretically capable of training multiple neural networks at the same time, this is, as far as we know, an untested problem. On the other hand, this could be implemented by using a distributed approach, where each sensor is responsible for part of the detection process. While this would have many benefits, like allowing the system to continue working even if some of the sensors are inactive, it also introduces all the challenges inherent to distributed systems.

Finally, another area of research that has not been explored significantly is the potential that CEP neuro-symbolic approaches have for explainability. Thanks to the human-understandable nature of the rules for the symbolic part of these approaches, they have the potential to generate explanations of why a certain decision was taken, based on those rules. As such, we believe that further research should explore how these systems can generate explanations that are easy to understand by humans. Furthermore, these explanations have the potential to make it easier for humans to identify when the system is wrong, and why. This could later be used to correct the system. If the mistake was caused by the perception layer, this could be fine-tuned to fix those erroneous cases. On the other hand, if the mistake was due to inaccurate rules, these could be modified to correctly take those cases into account.

6.4 Conclusion

Through the research shown in this thesis, we have demonstrated that neural networks and symbolic programming can be combined to provide an agile, reliable and efficient solution to complex event processing. More specifically, we have shown that defining rules specifying when complex events occur allows us to implement systems in an agile manner, with the possibility of changing their behaviour in a quick and easy manner.

We have also shown that neuro-symbolic approaches can be more reliable than neural-only approaches, as they can be implemented in a manner that ensures that complex events only happen when the conditions defined by the user are fulfilled. We also demonstrated that neuro-symbolic approaches can perform consistently in a range of situations wider than neural-only approaches can. Furthermore, we have evaluated the robustness of our neuro-symbolic approach, which has proved to be highly robust against many types of adversarial attacks in the form of poisoning.

Finally, with ProbCEP we have demonstrated that we can implement systems that detect complex events using proxy models. At the same time, we have also shown that DeepProbCEP can allow systems to train with very small amounts of data, making the approach very data-efficient. While it is true that approaches that combine neural networks with symbolic programming tend to be less time-efficient than neural-only approaches due to the time required to perform inference, we have shown that this problem can be reduced through the use of certain approaches, such as PreCompilation.

As such, we believe that approaches that combine neural networks with symbolic programming have great potential in the area of complex event processing.

Bibliography

[Abdulla, 2017] Abdulla, W. (2017). Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. `https://github.com/matterport/Mask_RCNN`.

[Akbari et al., 2020] Akbari, H., Palangi, H., Yang, J., Rao, S., Celikyilmaz, A., Fernandez, R., Smolensky, P., Gao, J., and Chang, S.-F. (2020). Neuro-symbolic representations for video captioning: A case for leveraging inductive biases for vision and language.

[Anicic et al., 2012a] Anicic, D., Rudolph, S., Fodor, P., and Stojanovic, N. (2012a). Real-time complex event recognition and reasoning-a logic programming approach. *Applied Artificial Intelligence - AAI*, 26:6–57.

[Anicic et al., 2012b] Anicic, D., Rudolph, S., Fodor, P., and Stojanovic, N. (2012b). Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3:397–407.

[Apriceno et al., 2021] Apriceno, G., Passerini, A., and Serafini, L. (2021). A Neuro-Symbolic Approach to Structured Event Recognition. In Combi, C., Eder, J., and Reynolds, M., editors, *28th International Symposium on Temporal Representation and Reasoning (TIME 2021)*, volume 206 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:14, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Besold et al., 2017] Besold, T. R., d'Avila Garcez, A. S., Bader, S., Bowman, H., Domingos, P. M., Hitzler, P., Kühnberger, K., Lamb, L. C., Lowd, D., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H., and Zaverucha, G. (2017). Neural-Symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902.

[Bezerra et al., 2021] Bezerra, E. D. C., Teles, A. S., Coutinho, L. R., and da Silva e Silva, F. J. (2021). Dempster-Shafer theory for modeling and treating uncertainty in IoT applications based on complex event processing. *Sensors*, 21(5).

[Board, 2019] Board, D. I. (2019). AI principles: Recommendations on the ethical use of artificial intelligence by the Department of Defense. *Supporting document, Defense Innovation Board*.

[Bruns et al., 2019] Bruns, R., Dunkel, J., and Offel, N. (2019). Learning of complex event processing rules with genetic programming. *Expert Systems with Applications*, 129:186–199.

[Burgueño et al., 2018] Burgueño, L., Boubeta-Puig, J., and Vallecillo, A. (2018). Formalizing complex event processing systems in maude. *IEEE Access*, 6:23222–23241.

[Carreira and Zisserman, 2017] Carreira, J. and Zisserman, A. (2017). Quo Vadis, action recognition? A new model and the Kinetics Dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308.

[Chapnik et al., 2021] Chapnik, K., Kolchinsky, I., and Schuster, A. (2021). DARLING: Data-aware load shedding in complex event processing systems. *Proc. VLDB Endow.*, 15(3):541–554.

[Cohen et al., 2018] Cohen, W. W., Yang, F., and Mazaitis, K. (2018). TensorLog: Deep learning meets probabilistic. *Journal of Artificial Intelligence Research*, 1.

[Cugola and Margara, 2012] Cugola, G. and Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3).

[Cunnington et al., 2022] Cunnington, D., Law, M., Lobo, J., and Russo, A. (2022). Inductive learning of complex knowledge from raw data. *arXiv preprint arXiv:2205.12735*.

[Cunnington et al., 2021a] Cunnington, D., Law, M., Russo, A., and Lobo, J. (2021a). FF-NSL: feed-forward neural-symbolic learner. *CoRR*, abs/2106.13103.

[Cunnington et al., 2021b] Cunnington, D., Law, M., Russo, A., Lobo, J., and Kaplan, L. (2021b). Towards neural-symbolic learning to support human-agent operations. In *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, pages 1–8.

[Dai and Muggleton, 2021] Dai, W.-Z. and Muggleton, S. (2021). Abductive knowledge induction from raw data. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International*

Joint Conference on Artificial Intelligence, IJCAI-21, pages 1845–1851. International Joint Conferences on Artificial Intelligence Organization.

[De Raedt and Kimmig, 2015] De Raedt, L. and Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47.

[De Raedt et al., 2007] De Raedt, L., Kimmig, A., and Toivonen, H. (2007). ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI*, pages 2462–2467.

[De Raedt et al., 2019] De Raedt, L., Manhaeve, R., Dumancic, S., Demeester, T., and Kimmig, A. (2019). Neuro-Symbolic = Neural + Logical + Probabilistic. In *Proceedings of the 2019 International Workshop on Neural- Symbolic Learning and Reasoning*, page 4.

[Demeester et al., 2016] Demeester, T., Rocktäschel, T., and Riedel, S. (2016). Lifted rule injection for relation embeddings. *EMNLP*.

[Diligenti et al., 2017] Diligenti, M., Gori, M., and Saccà, C. (2017). Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165. Combining Constraint Solving with Mining and Learning.

[Donadello et al., 2017] Donadello, I., Serafini, L., and d'Avila Garcez, A. S. (2017). Logic tensor networks for semantic image interpretation. *IJCAI*.

[Dostal, 2007] Dostal, B. C. (2007). Enhancing situational understanding through employment of unmanned aerial vehicle. *Army Transformation Taking Shape: Interim Brigade Combat Team Newsletter*, 01-18.

[Evans and Grefenstette, 2018] Evans, R. and Grefenstette, E. (2018). Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64.

[Fierens et al., 2015] Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., and De Raedt, L. (2015). Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(03):358–401.

[Gribble, 2001] Gribble, S. D. (2001). Robustness in complex systems. In *Proceedings eighth workshop on hot topics in operating systems*, pages 21–26. IEEE.

[Hara et al., 2018] Hara, K., Kataoka, H., and Satoh, Y. (2018). Can Spatiotemporal 3D CNNs retrace the history of 2D CNNs and ImageNet? In *CVPR*.

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (2017). Mask R-CNN. In *CVPR*.

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network.

[Hu et al., 2016] Hu, Z., Ma, X., Liu, Z., Hovy, E. H., and Xing, E. P. (2016). Harnessing deep neural networks with logic rules. *CoRR*, abs/1603.06318.

[Hudson and Manning, 2018] Hudson, D. A. and Manning, C. D. (2018). Compositional attention networks for machine reasoning. In *International Conference on Learning Representations*.

[Johnson et al., 2016] Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. (2016). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890.

[Kaplan et al., 2018] Kaplan, L., Cerutti, F., Sensoy, M., Preece, A., and Sullivan, P. (2018). Uncertainty Aware AI ML: Why and How. In *AAAI FSS-18: Artificial Intelligence in Government and Public Sector*.

[Khan and Curry, 2020] Khan, M. and Curry, E. (2020). Neuro-symbolic visual reasoning for multimedia event processing: Overview, prospects and challenges. In *CIKM (Workshops)*.

[Kimmig et al., 2011] Kimmig, A., Broeck, G. V. d., and Raedt, L. D. (2011). An algebraic Prolog for reasoning about possible worlds. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 209–214. AAAI Press.

[Law et al., 2020] Law, M., Russo, A., Bertino, E., Broda, K., and Lobo, J. (2020). FastLAS: Scalable inductive logic programming incorporating domain-specific optimisation criteria. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2877–2885.

[Law et al., 2021] Law, M., Russo, A., Broda, K., and Bertino, E. (2021). Scalable non-observational predicate learning in asp. In Zhou, Z.-H., editor, *Proceedings of the Thir-*

tieth International Joint Conference on Artificial Intelligence, IJCAI-21, pages 1936–1943. International Joint Conferences on Artificial Intelligence Organization.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2.

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755.

[Liu et al., 2018] Liu, K., Liu, W., Gan, C., Tan, M., and Ma, H. (2018). T-C3D: Temporal Convolutional 3D Network for real-time action recognition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

[Luckham, 2002] Luckham, D. C. (2002). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., USA.

[Manhaeve et al., 2018] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). DeepProbLog: Neural probabilistic logic programming. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Manhaeve et al., 2021] Manhaeve, R., Dumančiú, S., Kimmig, A., Demeester, T., and De Raedt, L. (2021). Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 298:103504.

[Mao et al., 2019] Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. (2019). The Neuro-Symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *CoRR*, abs/1904.12584.

[Margara et al., 2014] Margara, A., Cugola, G., and Tamburrelli, G. (2014). Learning from the past: Automated rule generation for Complex Event Processing. In *Proceedings of the*

8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, pages 47–58, New York, NY, USA. Association for Computing Machinery.

[Mascharka et al., 2018] Mascharka, D., Tran, P., Soklaski, R., and Majumdar, A. (2018). Transparency by design: Closing the gap between performance and interpretability in visual reasoning. *CoRR*, abs/1803.05268.

[Mehdiyev et al., 2015] Mehdiyev, N., Krumeich, J., Enke, D., Werth, D., and Loos, P. (2015). Determination of rule patterns in Complex Event Processing using machine learning techniques. *Procedia Computer Science*, 61:395–401. Complex Adaptive Systems San Jose, CA November 2-4, 2015.

[Minervini et al., 2017] Minervini, P., Demeester, T., Rocktäschel, T., and Riedel, S. (2017). Adversarial sets for regularising neural link predictors. *UAI*.

[Ministry of Defence UK, 2016] Ministry of Defence UK (2016). Understandig: Joint Doctrine Publication 04 (JDP 04). Ministry of Defence, UK.

[Mishra et al., 2018] Mishra, S., Jain, M., Siva Naga Sasank, B., and Hota, C. (2018). An ingestion based analytics framework for Complex Event Processing engine in Internet of Things. In Mondal, A., Gupta, H., Srivastava, J., Reddy, P. K., and Somayajulu, D., editors, *Big Data Analytics*, pages 266–281, Cham. Springer International Publishing.

[Oltramari et al., 2020] Oltramari, A., Francis, J., Henson, C., Ma, K., and Wickramarachchi, R. (2020). Neuro-symbolic architectures for context understanding.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. *NIPS Workshop*.

[Pearl, 2009] Pearl, J. (2009). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

[Preece et al., 2021] Preece, A. D., Braines, D., Cerutti, F., Furby, J., Hiley, L., Kaplan, L., Law, M., Russo, A., Srivastava, M., Vilamala, M. R., and Xing, T. (2021). Coalition situational understanding via explainable neuro-symbolic reasoning and learning. In Pham, T.

and Solomon, L., editors, *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, volume 11746, pages 453 – 464. International Society for Optics and Photonics, SPIE.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

[Robins, 2010] Robins, D. (2010). Complex event processing. In *Second International Workshop on Education Technology and Computer Science. Wuhan*, pages 1–10. Citeseer.

[Rocktäschel and Riedel, 2017] Rocktäschel, T. and Riedel, S. (2017). End-to-end differentiable proving. *NIPS*.

[Roig Vilamala et al., 2019] Roig Vilamala, M., Hiley, L., Hicks, Y., Preece, A., and Cerutti, F. (2019). A pilot study on detecting violence in videos fusing proxy models. In *2019 22th International Conference on Information Fusion (FUSION)*, pages 1–8.

[Roig Vilamala et al., 2020] Roig Vilamala, M., Taylor, H., Xing, T., Garcia, L., Srivastava, M., Kaplan, L., Preece, A., Kimmig, A., and Cerutti, F. (2020). A hybrid neuro-symbolic approach for complex event processing. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, 325.

[Roig Vilamala et al., 2021] Roig Vilamala, M., Xing, T., Taylor, H., Garcia, L., Srivastava, M., Kaplan, L., Preece, A., Kimmig, A., and Cerutti, F. (2021). Using DeepProbLog to perform Complex Event Processing on an audio stream. *StarAI*.

[Roig Vilamala et al., 2022] Roig Vilamala, M., Xing, T., Taylor, H., Garcia, L., Srivastava, M., Kaplan, L., Preece, A., Kimmig, A., and Cerutti, F. (2022). DeepProbCEP: A neuro-symbolic approach for complex event processing in adversarial settings. *Expert Systems with Applications*. Under review.

[Roldán et al., 2020] Roldán, J., Boubeta-Puig, J., Luis Martínez, J., and Ortiz, G. (2020). Integrating complex event processing and machine learning: An intelligent architecture for detecting IoT security attacks. *Expert Systems with Applications*, 149:113251.

[Sakama et al., 2021] Sakama, C., Inoue, K., and Sato, T. (2021). Logic programming in tensor spaces. *Annals of Mathematics and Artificial Intelligence*, 89.

[Salamon et al., 2014] Salamon, J., Jacoby, C., and Bello, J. P. (2014). A dataset and taxonomy for Urban Sound Research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA.

[Skarlatidis et al., 2015] Skarlatidis, A., Artikis, A., Filippou, J., and Paliouras, G. (2015). A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, 15(2):213–245.

[Sultani et al., 2018] Sultani, W., Chen, C., and Shah, M. (2018). Real-world anomaly detection in surveillance videos. In *CVPR*, pages 6479–6488.

[Susskind et al., 2021] Susskind, Z., Arden, B., John, L. K., Stockton, P., and John, E. B. (2021). Neuro-Symbolic AI: an emerging class of AI workloads and their characterization. *CoRR*, abs/2109.06133.

[Teymourian et al., 2012] Teymourian, K., Rohde, M., and Paschke, A. (2012). Knowledge-based processing of complex stock market events. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 594–597, New York, NY, USA. Association for Computing Machinery.

[Thys et al., 2019] Thys, S., Van Ranst, W., and Goedeme, T. (2019). Fooling automated surveillance cameras: Adversarial patches to attack person detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[Xing et al., 2020] Xing, T., Garcia, L., Vilamala, M. R., Cerutti, F., Kaplan, L., Preece, A., and Srivastava, M. (2020). Neuroplex: Learning to detect complex events in sensor networks through Knowledge Injection. *18th Conference on Embedded Networked Sensor Systems*, pages 489–502.

[Xing et al., 2019] Xing, T., Roig Vilamala, M., Garcia, L., Cerutti, F., Kaplan, L., Preece, A., and Srivastava, M. (2019). DeepCEP: Deep Complex Event Processing using distributed multimodal information. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 87–92.

[Xu et al., 2018] Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Broeck, G. (2018). A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR.

[Yadav, 2019] Yadav, P. (2019). High-performance complex event processing framework to detect event patterns over video streams. In *Proceedings of the 20th International Middleware Conference Doctoral Symposium*, Middleware '19, pages 47–50, New York, NY, USA. Association for Computing Machinery.

[Yadav et al., 2021] Yadav, P., Salwala, D., Sudharsan, B., and Curry, E. (2021). Gnosis-query-driven multimodal event processing for unstructured data streams. In *Proceedings of the 22nd International Middleware Conference: Demos and Posters*, Middleware '21, pages 16–17, New York, NY, USA. Association for Computing Machinery.

[Yankovitch et al., 2022] Yankovitch, M., Kolchinsky, I., and Schuster, A. (2022). HYPERSONIC: A hybrid parallelization approach for scalable complex event processing. *SIGMOD*.

Appendix A

Use of PreCompilation Library

The PreCompilation library allows the user to improve the speed of any ProbLog program that performs similar queries repeatedly by re-using the same AC, and thus removing the necessity of compiling the code each time. In order to do this, the user needs to define two classes, one for the input clauses and one for the queries. These classes should respectively inherit the classes `InputClause` and `Query` respectively, both defined in the library.

For the class inheriting from `InputClause`, the user needs to define two methods:

- `get_clause_format` should return a string indicating the format of the input clause in ProbLog code.
- `for_mock_model` should return a string indicating the format of the input clause that should be used to generate the mock model. Generally, this should be the clause in ProbLog format and with a probability of zero. This will cause ProbLog to generate any part of the AC that may be influenced by the corresponding input clause without modifying the value of the output.

For the class inheriting from `Query`, the user needs to define two methods, with an optional third if they want to perform recursive calls:

- `get_query_format` should return the format of the query in ProbLog.
- `update_result_timestamp` should return a Term (from the ProbLog library) that has the same shape as a provided term but with the timestamp value changed using the timestamp difference provided.

```python
class MyInputClause(InputClause):
    def get_clause_format(self):
        return '{probability}::{identifier}({timestamp}).\n'

    def for_mock_model(self):
        return self.to_problog_with(
            probability=self.zero_probability()
        )

class CoinQuery(Query):
    def get_query_format(self):
        return '\nquery({identifier}({timestamp})).'

    def update_result_timestamp(self, result, timestamp_diff):
        return Term(
            result.functor, Constant(result.args[0].functor + timestamp_diff)
        )

    def generate_feedback(self, evaluation, timestamp_diff):
        # Not required since we are not using feedback
        pass
```

Listing A.1: Example of classes implementing `InputClause` and `Query` for a scenario about flipping coins.

```python
class FeedbackQuery(Query):
    def get_query_format(self):
        return '\nquery({identifier}({timestamp})).'

    def update_result_timestamp(self, result, timestamp_diff):
        return Term(
            result.functor, Constant(result.args[0].functor + timestamp_diff)
        )

    def generate_feedback(self, evaluation, timestamp_diff):
        return [
            MyInputClause(
                identifier=clause.functor,
                timestamp=int(clause.args[0]) + timestamp_diff,
                probability=prob,
            )
            for clause, prob in evaluation.items()
        ]
```

Listing A.2: Example of classes implementing a `Query` allowing for a feedback option. This allows for recursive calls.

- `generate_feedback` should be implemented if we want to feed the results of the queries back into further calls. This should return a list of objects of a class inheriting from `InputClause`, which represent the values that come from the result of the queries.

Listing A.1 shows examples of how such classes can be defined to evaluate the result of a coin flip. Meanwhile, the class in Listing A.2 can be used in queries that require the result of the previous evaluation to find a new value. Please note that different classes can be combined when creating a precompilation.

Finally, the user needs to create an object of the class `PreCompilation`. In order to create such an object, the ProbLog code needs to be provided, together with the precompilation arguments. These precompilation arguments should include a list of the types of queries that the user may want to perform, as well as which input clauses are needed to perform those queries. It is worth noting that these input clauses should cover all possible combinations that may be encountered when performing the queries. This is because, otherwise, the system could output an incorrect prediction for any scenario that was not considered when generating these precompilation arguments. When created, the `PreCompilation` object will generate the ACs for all the queries given in the arguments. From then on, the `PreCompilation` object can be asked to perform any of those queries, which will return the output probabilities for these queries.

Appendix B

Sequence framework: A tool to detect patterns in streams of data

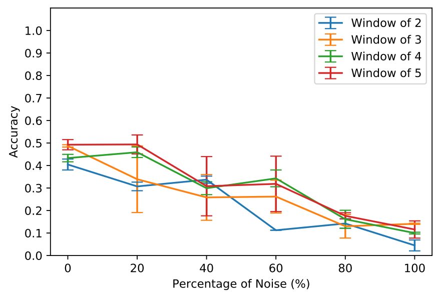
```
% Main interface for the framework.
sequence(L, W, T) :-
    reverse(L, L2), % Reverse list to simplify rule definitions
    sequenceEndingAt(L2, W, T).

% Add element X at the tail of the list
add_tail([], X, [X]).
add_tail([H | T], X, [H | L]) :- add_tail(T, X, L).

% The second list is the reverse of the first
reverse([], []).
reverse([X], [X]).
reverse([X | Xs], R) :-
    reverse(Xs, T),
    add_tail(T, X, R).

% This allows the use of exclude(X) to not allow certain simple
% events in the middle of the sequence
isNegation(exclude(X), X).

% Example of the structure of wraps. Added to prevent errors
% where wraps is not defined
wraps(thisShouldNotBeUsedWrapper, thisShouldNotBeUsedWrapped).

% Defining happensAt using atTime and wrapper
happensAt(X, T) :-
    atTime(T, A),
    wrapper(A, X).
```

```

% This allows the user to define how a keyword can wrap another
wrapper(X, N1) :- wraps(N1, N2), wrapper(X, N2).

% Check that at timestamp T none of the excluded events happen If
% there are no excluded, that is fine
checkExcluded(_, []).
% If there is an excluded, check that it does not happen at
% timestamp T, and then check the rest of excluded
checkExcluded(T, [X | E]) :-
    \+ happensAt(X, T),
    checkExcluded(T, E).

% An empty sequence will always be within if there are no simple
% events excluded
sequenceWithin([], [], _, _).

% An empty sequence will also be within if we have used all of
% the Remaining
sequenceWithin([], _, 0, _).

% An empty sequence will also be within if we reach the start of
% the timeline
sequenceWithin([], _, _, -1).

sequenceWithin([], E, Remaining, T) :-
    Remaining > 0,
    checkExcluded(T, E),
    NextRemaining is Remaining - 1,
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    sequenceWithin([], E, NextRemaining, Tprev).

% A sequence can be within Remaining of T if it starts at T
sequenceWithin(L, E, Remaining, T) :-
    sequenceEndingAt(L, Remaining, T).

sequenceWithin([X | L], E, Remaining, T) :-
```

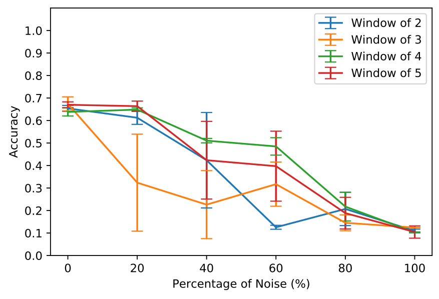
```
    isNegation(X, Y),
    sequenceWithin(L, [Y | E], Remaining, T).

% A sequence can be within Remaining of T if it is within
% NextRemaining of Tprev
sequenceWithin([X | L], E, Remaining, T) :-
    Remaining > 0,
    T >= 0,
    \+ isNegation(X, _),
    checkExcluded(T, E),
    NextRemaining is Remaining - 1,
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    sequenceWithin([X | L], E, NextRemaining, Tprev).

% A sequence will start at T and be within Remaining if X happens
% at T and the rest of the sequence is within NextRemaining of
% Tprev
sequenceEndingAt([X | L], Remaining, T) :-
    Remaining > 0,
    T >= 0,
    \+ isNegation(X, _),
    happensAt(X, T),
    NextRemaining is Remaining - 1,
    allTimeStamps(Timestamps),
    previousTimeStamp(T, Timestamps, Tprev),
    sequenceWithin(L, [], NextRemaining, Tprev).

sequenceEndingAt([X | L], Remaining, T) :-
    Remaining > 0,
    T >= 0,
    isNegation(X, _),
    sequenceWithin([X | L], [], Remaining, T).
```

Appendix C

Audio Robustness Results

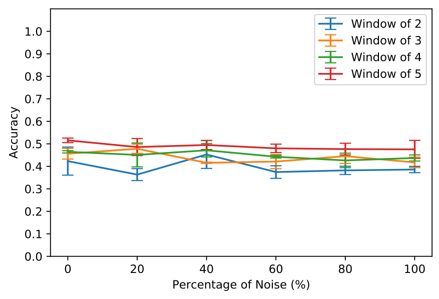


(a) Complex Event Accuracy for Urban Sounds 8K

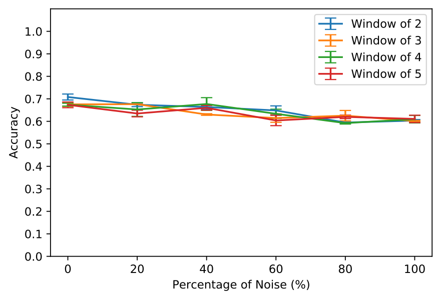


(b) Sound Accuracy for Urban Sounds 8K

Figure C.1: Accuracy results for Random noise.

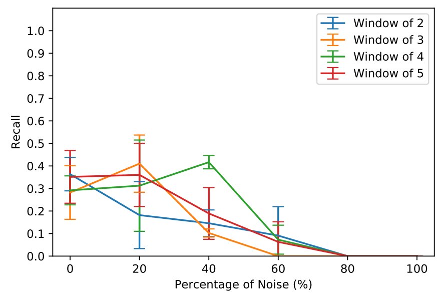


(a) Complex Event Accuracy for Urban Sounds 8K

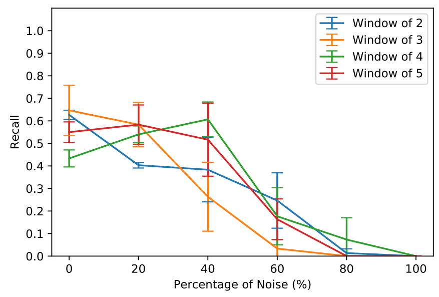


(b) Sound Accuracy for Urban Sounds 8K

Figure C.2: Accuracy results for Poisoning Single Label.

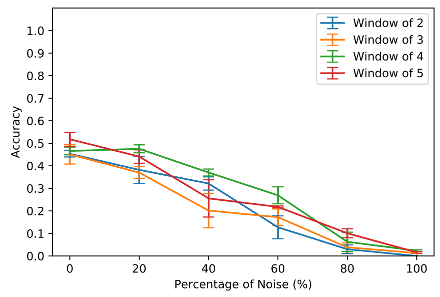


(a) Complex event recall in Urban Sounds

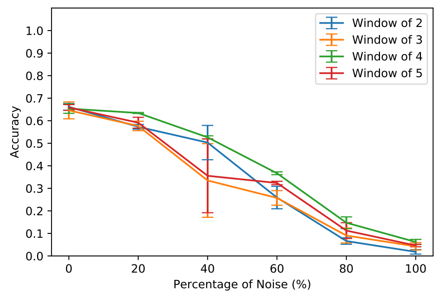


(b) Sound recall in Urban Sounds

Figure C.3: Recall results for the targeted simple and complex events under the Poisonin Single Label attack.

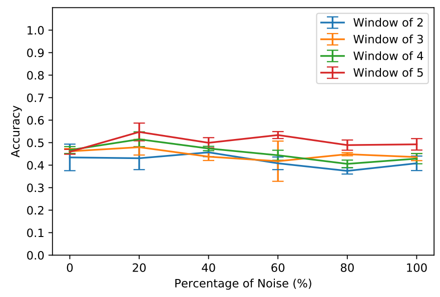


(a) Complex Event Accuracy for Urban Sounds 8K

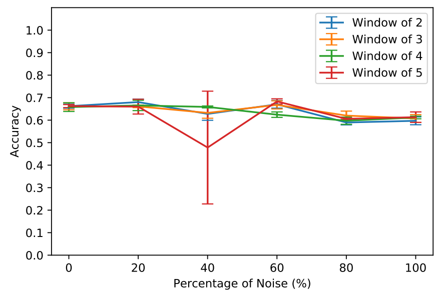


(b) Sound Accuracy for Urban Sounds 8K

Figure C.4: Accuracy results for Poisoning Multiple Labels.

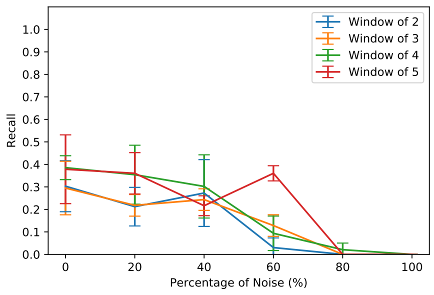


(a) Complex Event Accuracy for Urban Sounds 8K

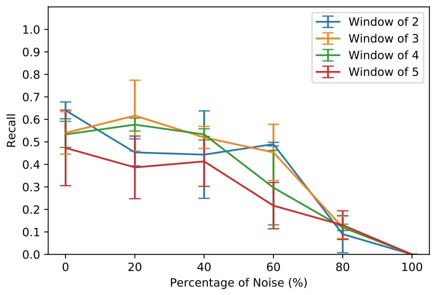


(b) Sound Accuracy for Urban Sounds 8K

Figure C.5: Accuracy results for Diluting.

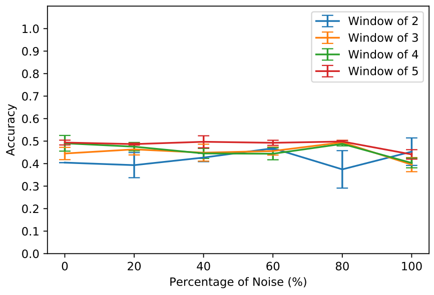


(a) Complex event recall in Urban Sounds

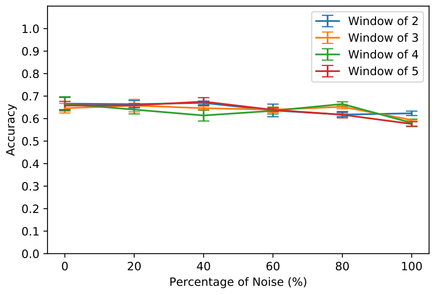


(b) Sound recall in Urban Sounds

Figure C.6: Recall results for the targeted simple and complex events under the Diluting attack.

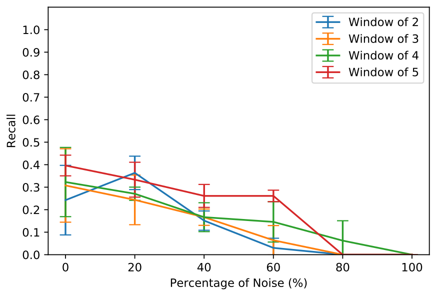


(a) Complex Event Accuracy for Urban Sounds 8K

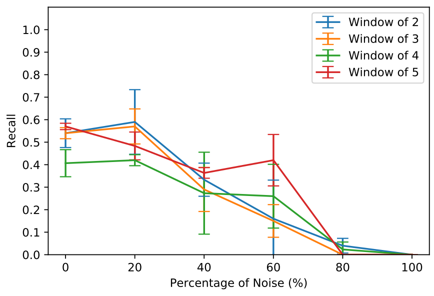


(b) Sound Accuracy for Urban Sounds 8K

Figure C.7: Accuracy results for Strict Diluting.



(a) Complex event recall in Urban Sounds



(b) Sound recall in Urban Sounds

Figure C.8: Recall results for the targeted simple and complex events under the Strict Diluting attack.

Appendix D

Glossary

Notation	Description
AC	An Arithmetic Circuit (AC) is the result of compiling a ProbLog logic program with a query. ACs can be efficiently evaluated, which outputs the prediction for the given query. However, the compilation process is comparatibly much slower..
adversarial attack	An attack performed against a machine learning algorithm with the aim of deteriorating the performance of said algorithm.
agile	Said of an approach that has demonstrated agility.
agility	Ability an approach to be implemented and modified easily, and allowing a wide range of options.
CEP	Complex event processing (CEP) systems are designed to identify situations of interest (called *complex events*) from an input stream of atomic data (where each atomic piece of information is called a *simple event*).
complex event	A situation of interest detected using a CEP approach.
data-efficient	Said of an approach that has demonstrated data-efficiency.
data-efficiency	Ability of an approach to train with small amounts of data.
efficiency	Ability of an approach to train with small amounts of data (called data-efficiency) and in a short time period (called time-efficiency).
efficient	Said of an approach that has demonstrated efficiency.
end-to-end training	Said of an approach that is able to train based on the input and output of the system, without requiring any intermediate labels.

Notation	**Description**
event	Sometimes used as shorthand for simple event. An atomic piece of information from an input stream.
high-level reasoning	One of the two sub-tasks of a neuro-symbolic system, responsible for determining the output of the system based on the symbolic information generated by the low-level perception sub-task. Implemented using a reasoning layer.
highly robust	Said of an approach regarding a type of poisoning attack where the accuracy of said approach does not decrease significantly for percentages of noise lower than 50%.
input stream	Used to refer to the source of data for a CEP approach.
knowledge distillation	A student-teacher approach where a neural network student is trained to emulate the behaviour of the teacher.
low-level perception	One of the two sub-tasks of a neuro-symbolic system, responsible for transforming the non-symbolic data into symbolic information, which will be used by the high-level reasoning sub-task. Implemented using a perception layer.
neural-only	An approach that relies solely on conventional neural networks.
neuro-symbolic	An approach that combines the use of neural networks with traditional rule-based systems, with the purpose of exploiting the benefits offered by both.
non-symbolic-focused	Used to refer to the types of CEP approaches designed to deal with non-symbolic data.
non-symbolic	Said of any type of data for which rules cannot (easily) be manually defined to extract relevant information.

Notation	**Description**
NRLogic model	A neural network model trained to emulate a logic layer using knowledge distillation.
perception layer	Part of a neuro-symbolic system responsible for transforming the non-symbolic input into symbolic information representing this input, which can then be used by the reasoning layer.
poisoning attack	Type of adversarial attack where the training data is targeted with the aim of deteriorating the performance of the systems trained with said data.
pre-trained	Used to refer to a neural network model which was trained prior to the integration into the current system.
proxy model	A model that is being used to approximate a task for which it was not intentionally designed or trained for.
reasoning layer	Part of a neuro-symbolic system responsible for applying a set of rules (which are usually manually defined) on symbolic information (extracted by the perception layer) to generate the output for the system.
reliability	Quality of an approach of being trustworthy and performing consistently well. Includes a consideration of the robustness of an approach.
reliable	Said of an approach that has demonstrated reliability.
robust	Said of an approach that has demonstrated robustness.
robustness	Ability of a system to continue to operate correctly across a wide range of operational conditions. Considered for approaches regarding how well they perform under adversarial attack.
simple event	An atomic piece of information from an input stream.

Notation	**Description**
sufficiently robust	Said of an approach regarding a type of poisoning attack where the accuracy of said approach does not decrease significantly for percentages of noise lower than 20%.
symbolic	Said of any type of data that is high-level (human-readable). Also used to refer to the types of approaches that deal with such data.
time-efficient	Said of an approach that has demonstrated time-efficiency.
time-efficiency	Ability of an approach to train quickly.